Constructive Membership Testing in Classical Groups

Elliot Mark Costi

September 2009

A thesis submitted to Queen Mary, University of London for the degree of Doctor of Philosophy

School of Mathematical Sciences,

Queen Mary, University of London,

Mile End Road,

E1 4NS

Declaration

I declare that, unless stated otherwise, all the work in this thesis is my own original work.

Signed: E. Get

Abstract

Let G be a perfect classical group defined over a finite field F and generated by a set of standard generators X. Let E be the image of an absolutely irreducible representation of G by matrices over a field of the natural characteristic. Given the image of X in E, we present algorithms that write an arbitrary element of E as a straight-line programme in this image of X in E. The algorithms run in polynomial time.

Contents

1 Introduction 6

1.1 Background 6

1.2 Motiviation 7

1.3 The main results 8

1.4 Further Machinery 8

1.5 Notation 9

2 The natural representation 10

2.1 The generating sets 10

2.1.1 How each generator embeds into its respective classical group . . 14

2.2 SL(d, q) in its natural representation 15

2.2.1 Pseudo-code 18

2.2.2 Complexity 25

2.3 Sp(d, q) in its natural representation 26

2.3.1 Introduction 26

2.3.2 Description of the Method 27

2.3.3 Performing this Method Using the Generating Set 28

2.3.4 Pseudo-code 30

2.4 SU(d, q) in its natural representation, d even 44

2.5 SU(d, q) in its natural representation, d and q odd 48

2.5.1 The main algorithm 48

2.5.2 SU$(3, q)$, q odd 51

2.6 $\mathrm{SU}(d,q)$ in its natural representation, d odd and q even 55
2.6.1 The generating set 55
2.6.2 The embedding of $\mathrm{SU}(d-1,q)$ into $\mathrm{SU}(d,q)$ 57
2.6.3 $\mathrm{SU}(3,q)$, q even 59
2.7 $\Omega^+(d,q)$ in its natural representation, d even 60
2.7.1 Killing the final 4×4 block of a matrix 62
2.7.2 $\mathrm{SO}^+(d,q)$ 63
2.8 $\Omega^-(d,q)$ in its natural representation, d even and q odd 63
2.8.1 Introduction 63
2.8.2 Description of the Method 66
2.8.3 $\mathrm{SO}^-(d,q)$ 69
2.9 $\Omega^-(d,q)$ in its natural representation, d and q even 69
2.9.1 Forming the Generating Set 69
2.9.2 The Embedding of $\Omega^+(d-2,q)$ into $\Omega^-(d,q)$ 73
2.10 $\Omega(2d+1,q)$ in its natural representation, q odd 75
2.10.1 $\mathrm{SO}(2d+1,q)$ 77

3 Non-natural representations in the defining characteristic 79
3.1 Introduction 79
3.2 $\mathrm{SL}(d,q)$ in a non-natural representation 80
3.2.1 Constructing ϕ, the map from the natural to the non-natural representation 80
3.2.2 Reducing the pre-image of g by one dimension 81
3.2.3 The action of the p-group on the reduced matrix g 83
3.2.4 Worked Example on the Exterior Square 85
3.2.5 Pseudo-code 90
3.2.6 Complexity 98
3.2.7 Testing 98
3.3 $\mathrm{Sp}(d,q)$ in a non-natural representation 99

3.4 SU(d,q) in a non-natural representation, d even 101
3.5 SU(d,q) in a non-natural representation, d odd 104
3.6 $\Omega^+(d,q)$ in a non-natural representation 106
3.7 $\Omega^-(d,q)$ in a non-natural representation 109
3.8 $\Omega^\circ(d,q)$ in a non-natural representation 114

4 Modifying Ruth Schwingel's second algorithm: `Ruth2` 115
4.1 Introduction 115
4.2 Extending the algorithm to cope with prime power fields 116
4.2.1 Defining the partial ordering and examples 117
4.3 Defining the correct chief series 118
4.3.1 Example - canonising a vector over $\mathbb{F}_{7^3}$ 120
4.3.2 Example - canonising a subspace over $\mathbb{F}_{7^3}$ 121
4.4 Complexity 123
4.4.1 Timings 124

5 An algorithm to write an element of any unipotent matrix group as a word in its generating set 126
5.1 Introduction 126
5.2 Finding a suitable chief series (`Algorithm 32`) 127
5.2.1 Pseudo-code 129
5.3 Description of the main algorithm 131
5.4 Pseudo-code 136
5.4.1 Complexity 139
5.4.2 Timings 139

6 An implementation 141
6.1 The Natural Representation 141
6.2 Non-Natural Representations 144

Acknowledgements

First and foremost, I would like to thank my supervisor Charles Leedham-Green for his ongoing advice and suggestions, for helping me secure funding and for allowing me to undertake research in a field that I enjoyed. I would also like to thank Eamonn O'Brien for his advice, for helping with the implementation of the algorithms and for giving me the opportunity to study for three months at Auckland. In addition to this, I would like to thank Alan Camina and the QMUL mathematics department for providing the funding that allowed me to undertake research overseas.

I acknowledge the Mathematics and Computer Science departments for the interdisciplinary grant that made the past three years possible.

Chapter 1

Introduction

1.1 Background

Computational Group Theory is an area of mathematics where group theoretical problems, often too cumbersome to be performed by hand, are implemented as algorithms in packages such as GAP and MAGMA. Within this field, the Matrix Recognition Project is a long-running international research project whose aim is to produce efficient algorithms for solving problems involving matrix groups over finite fields, as well as making efficient implementations of these algorithms.

Generally, a classical group is a matrix group over a finite field, either of special linear type, or preserving a bilinear, sesquilinear or quadratic form on the vector space on which it acts. For the purposes of this thesis, a classical group is one of the following: $\mathrm{SL}(d,q)$, where $d > 1$; $\mathrm{Sp}(d,q)$, where d is even; $\mathrm{SU}(d,q)$, where $d > 2$; $\mathrm{SO}^+(2d,q)$, where $d > 1$; $\mathrm{SO}^-(2d,q)$, where $d > 1$; $\mathrm{SO}^\circ(2d+1,q)$, where $d > 0$ and q is odd; $\Omega^+(2d,q)$, where $d > 1$; $\Omega^-(2d,q)$, where $d > 1$; $\Omega^\circ(2d+1,q)$, $d > 0$ and q is odd. Each classical group will preserve a specific form as given in a paper by Charles Leedham-Green and Eamonn O'Brien [4]. For a fuller explanation of how these groups arise see both [2] and [7]. These groups are perfect, with exception of the special orthogonal groups $\mathrm{SO}^+(2d,q)$, $\mathrm{SO}^-(2d,q)$ and $\mathrm{SO}^\circ(2d+1,q)$; and ten groups of small order: $\mathrm{SL}(2,2) = \mathrm{Sp}(2,2), \mathrm{SL}(2,3) = \mathrm{Sp}(2,3), \mathrm{Sp}(4,2), \mathrm{SU}(3,2), \Omega(3,2), \Omega(3,3), \Omega^+(4,2)$ and $\Omega^+(4,3)$. In

this thesis, we shall be looking at absolutely irreducible representations of these groups in the natural characteristic. That is to say, homomorphisms from classical groups onto irreducible subgroups of $\mathrm{GL}(n, q')$, where q and q' are both powers of the same prime.

1.2 Motiviation

A fundamental problem is the explicit membership problem. Given a subset X of some universal group U and an element $g \in U$, determine whether or not $g \in G = \langle X \rangle$ and, if $g \in G$, return a straight-line programme (SLP) in X that evaluates to g.

In this thesis, U will be $\mathrm{SL}(n, q')$ and:

- G will be isomorphic to a central quotient of a known classical group;
- X will correspond to a set of canonical generators of the classical group as defined in Chapter 2.

We conclude this section with a definition of a straight-line programme (SLP). One may intuitively think of an SLP for $g \in G = \langle X \rangle$ as an efficiently stored group word on X that evaluates to g. An SLP is a data structure for words that ensures that subwords occurring multiple times are computed only once.

Definition 1.2.1 *Given a set of generators X, an SLP is a sequence $(s_1, s_2, \ldots, s_n)$ where each s_i represents one of the following:*

- *an $x \in X$;*
- *product $s_j s_k$, where $j, k < i$;*
- *a power s_j^n, where $j < i$ and $n \in Z$;*
- *a conjugate $s_j^{s_k}$ where $j, k < i$.*

1.3 The main results

In this thesis, we describe algorithms to do the following. Take a classical group G defined over a finite field F of characteristic p and generated by a set of standard generators. Now take an arbitrary element $g \in G$. We first describe algorithms that will take as input g and the name of G and return a straight-line programme (SLP) in the given generating set that evaluates to g. In Chapter 3, we take a non-natural representation $E < \mathrm{SL}(n, q')$ of G, where E is in the natural characteristic, i.e. q' is a power of p. E will be isomorphic to a central quotient of G; that is to say G possibly modulo some subset of the set of scalar matrices. Given the image of the standard generating set in E, we present algorithms that decide whether an arbitrary element of $\mathrm{SL}(n, q')$ is in E and if so, write this element as a straight-line programme in the image of this standard generating set. The algorithms are deterministic and run in polynomial time.

Within the wider scope of the Matrix Recognition Project, many other individuals are involved in research to implement similar algorithms for other simple groups. The algorithms outlined in Chapter 3 are the second part to a two-stage process. The first stage of the process is as follows. Given a set of matrices X that generate a group E, it must first be decided if E is an absolutely irreducible non-natural representation in the same characteristic of a classical group G. If so, then we wish to write the image of the standard generating set of G in E in terms of the set X. Algorithms to complete this initial stage of the process are currently in production.

1.4 Further Machinery

In order to complete the above tasks, some further machinery needs to be introduced. Hence in Chapter 4, we look at an algorithm originally written by Ruth Schwingel [6] that takes as input a unipotent matrix group K over a prime field and a subspace U of the natural vector space on which K acts. The algorithm then returns the following: a

canonical element $\bar{U}$ of the orbit of U under K; an element $k \in K$ such that $U^k = \bar{U}$; generators for the stabiliser of U in K. The original implementation of this algorithm was only designed to work over matrices with entries in prime fields and so we provide an implementation that is able to take as input matrix groups written over a field of prime power order. We also look at ways of improving the efficiency of this algorithm in other aspects.

In Chapter 5, we describe an algorithm that takes as input a unipotent matrix group $K < \mathrm{GL}(d, q)$ and an element $Y \in \mathrm{GL}(d, q)$ and tests constructively for membership of Y in K. If Y is in K, then an SLP is returned in the user-defined generating set of K.

In all chapters, we provide a complexity analysis of the algorithms and provide timings of their implementations in MAGMA. The complexity will be measured in the order of the number of field operations needed for the algorithm to complete.

1.5 Notation

We now clarify the notation that is used in this thesis.

- Whenever an element g of a group G generated by a set X is given, then $\bar{g}$ denotes an SLP for g written in X.
- Let $F = \mathrm{GF}(p^e)$ and let ω be a primitive element of F. If $\alpha \in F$ is an arbitrary element, then α can be written as a polynomial of degree at most $e-1$ over ω. If we wish to refer to the coefficient of ω^r in α, then we denote this by α_r.

Chapter 2

The natural representation

2.1 The generating sets

The generators of each classical group are given below, with the exception of the orthogonal groups in characteristic 2 and the unitary groups in odd dimension and even characteristic, which are given later. The generators that we will be using are given in the tables on the following page in a reduced form. That is to say, if a generator is of the following form: $\begin{pmatrix} A & 0 \\ 0 & I \end{pmatrix}$, where I represents an identity matrix and 0 represents a zero matrix, we only exhibit each generator below as A. An explanation of how the elements of each generating set embed into the matrix group is given after the tables.

Each classical group is written with respect to a basis consisting of hyperbolic pairs. The form that each matrix group preserves is discussed in the section for that particular classical group. However, we describe here the basis that each matrix group acts on.

1. For $\mathrm{SL}(n,q)$, any basis is hyperbolic as the group does not preserve a classical form.

2. For $\mathrm{Sp}(2n,q)$, $\mathrm{SU}(2n,q)$ and $\Omega^+(2n,q)$, the basis is a set of n hyperbolic pairs (e_i, f_i) and is ordered thus: $\{e_1, f_1, \ldots, e_n, f_n\}$.

3. For $\mathrm{SU}(2n+1,q)$, the basis is formed of n hyperbolic pairs (e_i, f_i) plus an element w of norm 1 that generates a one-dimensional space that does not contain any

isotropic vectors with respect to a sesquilinear form: $\{e_1, f_1, \ldots, e_n, f_n, w\}$.

4. For $\Omega(2n+1, q)$, the basis is formed of n hyperbolic pairs (e_i, f_i) plus an element w of norm 1 that generates a one-dimensional space that does not contain any singular vectors with respect to a quadratic form: $\{e_1, f_1, \ldots, e_n, f_n, w\}$.

5. For $\Omega^-(2n, q)$, q odd, the basis is formed of $(n-1)$ hyperbolic pairs (e_i, f_i) plus two elements w_1 and w_2 that generate a two-dimensional space that does not contain any singular vectors with respect to a quadratic form: $\{e_1, f_1, \ldots, e_n, f_n, w_1, w_2\}$, where the norm of w_1 is -2, the norm of w_2 is 2ω and $w_1.w_2 = 0$.

In all but one case, we describe v as a signed permutation matrix acting on the hyperbolic basis for V. We adopt the following notation. Given a basis for V, a signed permutation matrix with respect to this basis will be given as a product of disjoint signed cyclic permutations of the basis elements. Such a cycle either permutes the vectors in the cycle, no sign being involved, or it sends each vector in the cycle to the next, except for the last vector which is sent to minus the first vector. In this case the cycle is adorned with the superscript $-$, as in $(e_1, e_2, \ldots, e_n)^-$. The superscript $+$ has no effect, so that $(e_1, e_2, \ldots, e_n)^+ = (e_1, e_2, \ldots, e_n)$. If we use the notation $(e_1, e_2, \ldots, e_n)^{\epsilon_n}$, then $\epsilon_n = +$ if n is odd, and $\epsilon_n = -$ if n is even.

In the following tables, ω is a fixed primitive element of $\mathrm{GF}(q)$, except for the unitary groups where ω is a fixed primitive element of $\mathrm{GF}(q^2)$. For the unitary groups defined over the field $\mathrm{GF}(q^2)$, $\alpha = \omega^{(q+1)/2}$ in odd characteristic. For $\Omega^-(2n, q)$, let γ be a fixed primitive element of $GF(q^2)$ such that $\gamma^{q+1} = \omega$. Then the variables A, B and C given in the definition have the following values, with α defined as for the unitary groups:

$$\begin{aligned} A &= \frac{1}{2}(\gamma^{q-1} + \gamma^{-q+1}) \\ B &= \frac{1}{2}\alpha(\gamma^{q-1} - \gamma^{-q+1}) \\ C &= \frac{1}{2}\alpha^{-1}(\gamma^{q-1} - \gamma^{-q+1}). \end{aligned}$$

Group	s	t	δ	u	v	x	y
$\mathrm{SL}(n,q)$	$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 \\ 0 & \omega^{-1} \end{pmatrix}$	I_2	$\begin{pmatrix} 0 & 1 \\ -I_{n-1} & 0 \end{pmatrix}$ or $\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix}$, $d=3$	I_4	I_4
$\mathrm{Sp}(2n,q)$	$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 \\ 0 & \omega^{-1} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$(e_1, e_2, \ldots, e_n)(f_1, f_2, \ldots, f_n)$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$	I_4
$\mathrm{SU}(2n,q)$	$\begin{pmatrix} 0 & \alpha \\ \alpha^{-q} & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & \alpha \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega^{q+1} & 0 \\ 0 & \omega^{-(q+1)} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$(e_1, e_2, \ldots, e_n)(f_1, f_2, \ldots, f_n)$	$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 & 0 & 0 \\ 0 & \omega^{-q} & 0 & 0 \\ 0 & 0 & \omega^{-1} & 0 \\ 0 & 0 & 0 & \omega^{q} \end{pmatrix}$
$\mathrm{SU}(2n+1,q)$ q, odd	$\begin{pmatrix} 0 & \alpha \\ \alpha^{-q} & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & \alpha \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega^{q+1} & 0 \\ 0 & \omega^{-(q+1)} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$(e_1, e_2, \ldots, e_n)(f_1, f_2, \ldots, f_n)$	$\begin{pmatrix} 1 & -1/2 & 1 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 & 0 \\ 0 & \omega^{-q} & 0 \\ 0 & 0 & \omega^{q-1} \end{pmatrix}$

Table 2.1: Standard generators for non-orthogonal classical groups

Group	s	t	δ	u	v
$\Omega^+(2n,q)$	$\begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 & 0 & 0 \\ 0 & \omega^{-1} & 0 & 0 \\ 0 & 0 & \omega & 0 \\ 0 & 0 & 0 & \omega^{-1} \end{pmatrix}$	I_4	$(e_1, e_2, \ldots, e_n)^{\epsilon_n}(f_1, f_2, \ldots, f_n)^{\epsilon_n}$
	s'	t'	δ'		
	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 & 0 & 0 \\ 0 & \omega^{-1} & 0 & 0 \\ 0 & 0 & \omega^{-1} & 0 \\ 0 & 0 & 0 & \omega \end{pmatrix}$		
Group	t	t'	δ	u	v
$\Omega^-(2n,q)$ q, odd	$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 & 0 & 0 \\ 0 & \omega^{-1} & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & C & A \end{pmatrix}$	$(e_1, e_2)^-(f_1, f_2)^-$	$(e_1, \ldots, e_{n-1})^{\epsilon_{n-1}}(f_1, \ldots, f_{n-1})^{\epsilon_{n-1}}$
Group	s	t	δ	u	v
$\Omega(2n+1,q)$ q, odd	$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega^2 & 0 & 0 \\ 0 & \omega^{-2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$	I_4	$(e_1, \ldots, e_n)^{\epsilon_n}(f_1, \ldots, f_n)^{\epsilon_n}$

Table 2.2: Standard generators for orthogonal groups

2.1.1 How each generator embeds into its respective classical group

1. For $\mathrm{SL}(n,q)$, each generator given in the table above embeds itself into the matrix group by sitting in the top left-hand corner of an $n \times n$ matrix. For example, the generator s above will become:

$$\begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ -1 & 0 & 0 & \dots & 0 \\ 0 & 0 & & & \\ & \vdots & & I_{n-2} & \\ 0 & 0 & & & \end{pmatrix}.$$

2. For $\mathrm{Sp}(2n,q)$, each generator sits in the top left-hand corner of a $2n \times 2n$ matrix, with the exception of the generator x, which sits in the bottom right-hand corner.

3. For $\mathrm{SU}(2n,q)$, each generator sits in the top left-hand corner of a $2n \times 2n$ matrix, with the exception of the generators x and y, which sit in the bottom right-hand corner.

4. For $\mathrm{SU}(2n+1,q)$, each generator sits in the top left-hand corner of a $(2n+1) \times (2n+1)$ matrix, with the exception of the generators x and y, which sit in the bottom right-hand corner. Note that generator v fixes the basis vector w.

5. For $\Omega^+(2n,q)$, each generator sits in the top left-hand corner of a $2n \times 2n$ matrix.

6. For $\Omega^-(2n,q)$, the generators u and v sit in the top left-hand corner of a $2n \times 2n$ matrix, and the generators t, t' and δ sit in the bottom-right hand corner.

7. For $\Omega(2n+1,q)$, each generator sits in the bottom right-hand corner of a $2n \times 2n$ matrix, with the exception of v, which sits in the top left-hand corner.

2.2 $\mathrm{SL}(d, q)$ in its natural representation

We first consider the simplest case: when the classical group $G = E$ is $\mathrm{SL}(d, q)$. Let A be an arbitrary element of $\mathrm{SL}(d, q)$. By applying row and column operations to A, it can be reduced to the identity matrix. We first consider the row and column operations necessary to do this and then consider how to perform these row and column operations by multiplying A by elements of the generating set.

In this thesis, we will use the term *killed* to mean setting an entry of a matrix or vector to zero. If we are talking about a row of a matrix, the term *killed* will mean making every entry zero, with the exception of one of the entries being a 1. If we are talking about an entire matrix, the term *killed* will mean either using row and column operations or matrix multiplications, to reduce it to the identity.

1. Add a multiple of one row to the top to get a 1 in the (1, 1) entry of A. If the (1, 1) entry of A is the only non-zero entry in the first column, then this will not be possible. In this case, we first add the second column to the first, thus creating other non-zero entries in the first column.

2. Once A has a 1 in the (1, 1) entry, add a suitable multiple of the first row / column of A to every other row / column until every other entry in the first row / column of A are all zero.

3. Move the top row of A to the bottom and the first column of A to the far right of the matrix.

4. Recursively repeat this process until every column and row of A has been killed. A is now the identity matrix.

We now discuss how to perform these row and column operations using the matrices in the generating set.

1. Post-multplying A by the element t adds the first column of A to the second. Pre-multiplying A by $(t^s)^{-1}$ adds the first row of A to the second.

2. If q is odd, the generators s and v generate S, a monomial subgroup of index 2 in $C_2 \wr S_d$, and S maps surjectively onto S_d. If q is even, s and v generate S_d. Hence, using s and v, the rows / columns of the matrix A can be permuted so that row i can be moved to row 1 or column j to column 1.

3. Various combinations of conjugates of t by powers of δ can be used to add a suitable multiple of any row to the first in order to get a 1 in the (1, 1) entry. See the lemma below for details of how these conjugates are formed.

4. Similarly, any multiple of the first row/column of A can be added to every other row/column. Hence, it is possible to use the generating set make all the entries in the top row and first column zero, with exception of the (1, 1) entry.

5. Conjugating A by v will move the top row of A to the bottom and the first column to the last, possibly with some negation of the entries.

6. By working through the matrix A recursively, A is reduced to the identity matrix. Killing the k-th row and column does not effect the rows and columns that have already been dealt with due to the fact that the first row and column will contain zeroes in the $(d-k+1)$-th to d-th entries.

7. By keeping track of the matrix multiplications performed, we get that $x_1 \dots x_r A x_{r+1} \dots x_s = I_d$, where the x_i represent elements of the generating set. Hence, this equation can be rearranged to get A in terms of the generating set and we are done.

When in the prime power case, an entry of A is cleared by considering this entry as a polynomial over ω with coefficients in the prime field. Each coefficient is then killed using transvections of the form:

$$\begin{pmatrix} 1 & \omega^i & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & & & \\ & \vdots & & \mathrm{I}_{d-2} & \\ 0 & 0 & & & \end{pmatrix},$$

where $0 \leq i \leq e-1$ and e is the degree of the field. In the following lemma, we talk of writing ω in terms of even powers of itself. In practical implementations of the algorithm, we use the MAGMA package to do this by coercing ω into a field that has ω^2 defined as its generating element. The following lemma shows that this is always possible.

Lemma 2.2.1 *The transvections as described above can be constructed using conjugates of t by δ.*

PROOF: An easy calculation shows that $t^{\delta^{-1}}$ gives the transvection with ω^2 in the (1, 2) slot and similarly, $t^{\delta^{-i}}$ gives the transvection with ω^{2i} in the (1, 2) slot. Multiplying $t^{\delta^{-i}}$ by $t^{\delta^{-j}}$ gives $\omega^{2i} + \omega^{2j}$ in the (1, 2) slot.

Suppose the field that we are working over is F. Now consider the subfield K of F generated by ω^2. As K contains all even powers of ω and 0, it's size is at least $\frac{|F|-1}{2}+1$. Therefore, the size of K is larger than half of the size of F and so $F = K$. Hence, ω can be written as a sum of even powers of itself and so the matrix with ω in the (1, 2) slot can be formed by multiplying $t^{\delta^{-i}}$ for various i. Call this element O.

A calculation shows that $O^{\delta^{-i}}$ gives the transvection with ω^{2i+1} in the (1, 2) slot and hence the odd powers of ω can also be formed by conjugates of t by δ. □

By consdering ω as an element of K, as defined above, we calculate ω as a sum of even powers of itself so that the required matrices can be formed.

Rather than perfoming the matrix multiplications, the algorithm instead replaces each matrix multiplication with a row or column operation that would have the same

result. This greatly reduces both the complexity of the algorithm and the time taken for the algorithm to complete.

The algorithm also makes a note of each matrix multiplication as it happens in the form of a straight-line programme (SLP) and returns the element A as an SLP in the generators of $\mathrm{SL}(d, q)$.

2.2.1 Pseudo-code

We now summarise this algorithm in pseudo-code by breaking each part of the algorithm into smaller functions and encapsulating them into the main function `SLWordInGen`. In the functions that follow, the SLPs corresponding to the standard generating set are global variables.

We first describe some simpler functions that will not require a fuller explanation to be given:

1. The function `AddColumn`(A, n, i, j) takes as input a matrix A and adds n times column i to column j;
2. The function `AddRow` is the obvious row equivalent;
3. `IsEven` takes an integer as its input and decides whether it is even or not;
4. `IsOdd` is the obvious odd integer equivalent.
5. `Transpose`(A) returns the transpose of an input matrix A.
6. `ZeroMatrix`(F, d, d) returns a zero matrix of size $d \times d$ over the field F.

We remind the reader that the notation used in these algorithms is as follows:

1. A bar over an element of the generating set in this algorithm signifies that this is the equivalent element as an SLP.

2. If β is a field element, then we denote β_r to be the coefficient of ω^r considered as an integer, where β is written as a polynomial of degree at most $e-1$ over the primitive element ω, with coefficients in the prime field. For the (i,j)-th element of a matrix A, this will be denoted $A_{i,j,r}$. If the weight of a matrix is (j_0, j_1, j_2) (see Definition 4.2), then $A_{[j_0,j_1,j_2]}$ denotes the coefficient of ω^{j_2-1} in the (j_0, j_0+j_1)-th entry of the matrix A.

3. $\bar{O}$ is the SLP that corresponds to the transvection with ω in the (1, 2) entry and is formed by applying the method to create the corresponding transvection, as described above, to the generators of the SLP group.

Algorithm 1: SLWordInGen(G, A)

/* $G = \mathrm{SL}(d, q)$ for some d and q. A is an element of G. Return the identity element of G, to show that the entries in A have been fully killed, and a word in the standard generating set for G written as an SLP. Suppose that $\hat{A}$ is A as it is at the start of the algorithm: unmodified. The two SLPs s_1 and s_2 are modified throughout the algorithm so that the equation $\hat{s_1}\hat{A}\hat{s_2} = A$ always holds, where $\hat{s_i}$ is the evaluation of s_i on the generating set. */

1 **begin**

2 assert Determinant$(A) = 1$;

3 (s_1, s_2) :=(identity slp, identity slp);

4 **for** $i \in \{1, \ldots, d-1\}$ **do**

5 $A, s_1, s_2 :=$ GetAOne(A, s_1, s_2);

6 $A, s_1 :=$ RowOp$(A, s_1, i-1)$;

7 $A, s_2 :=$ ColOp$(A, s_2, i-1)$;

8 $A, s1, s2 :=$ ConjByV$(A, s1, s2)$;

9 $k := k + 1$;

10 **end**

11 return A, $s_1^{-1}s_2^{-1}$;

12 **end**

Algorithm 2: GetBetaTransvection(β, i)

/* β is an element of $GF(q)$. Return a transvection T of $\mathrm{SL}(d,q)$ as an SLP with β as its (1, i) entry. */

1 begin

2 $T := (\bar{t}^{-1})^{\beta_0}$;

3 for $r \in \{2, \ldots, e\}$ do

4 if IsEven(r) then

5 $T := T((\bar{O}^{-1})^{\theta})^{\beta_{r-1}}$, where $\theta := \bar{\delta}^{-\frac{r-2}{2}}$;

6 else

7 $T := T((\bar{t}^{-1})^{\theta})^{\beta_{r-1}}$, where $\theta := \bar{\delta}^{-\frac{r-1}{2}}$;

8 end

9 end

10 end

11 $T := T^{\pi}$, where $\pi := (\bar{v}\bar{u})^{i-2}$;

12 return T;

13 end

Algorithm 3: GetAOne(A, s_1, s_2)

/* A is an element of SL(d, q) for some d and q. s_1 and s_2 are SLPs in the aforementioned generating set X. Return the matrix A, modified by having its (1, 1) entry set to 1 and the corresponding modified SLPs s_1 and s_2. */

1 begin
2 if $A_{1,1} = 1$ then
3 return A, s_1, s_2;
4 end
5 if $\forall i \in \{2, \ldots, d\}, A_{i,1} = 0$ then
6 $A :=$ AddColumn$(A, -1, 2, 1)$;
7 $s_2 := s_2 \overline{t^u}$;
8 end
9 $i := \min\{2, \ldots, d : A_{i,1} \neq 0\}$;
10 $\beta := \frac{A_{1,1}-1}{A_{i,1}}$;
11 $A :=$ AddRow$(A, -\beta, i, 1)$;
12 $T :=$GetBetaTransvection(β, i);
13 $s_1 := Ts_1$;
14 return A, s_1, s_2;
15 end

Algorithm 4: RowOp(A, S, k)

/* A is an element of $SL(d, q)$ for some d and q. S is an SLP in the standard generating set. k is an integer - the number of rows and columns that have already been killed. Return A with its first column killed and the corresponding modified SLP S. */

1 begin
2 for $j \in \{2, \ldots, d-k\}$ do
3 $\pi := \bar{u}(\overline{vu})^{j-2}$;
4 for $r \in \{1, \ldots, e\}$ do
5 if IsEven(r) then
6 $S := (\bar{O}^{\theta})^{A_{j,1,r}} S$, where $\theta := \bar{\delta}^{-\frac{r-2}{2}} \pi$;
7 AddRow$(A, -\omega^{r-1} A_{j,1,r}, i, j)$;
8 end
9 if IsOdd(r) then
10 $S := (\bar{t}^{\theta})^{A_{j,1,r}} S$, where $\theta := \bar{\delta}^{-\frac{r-1}{2}}) \pi$;
11 AddRow$(A, -\omega^{r-1} A_{j,1,r}, i, j)$;
12 end
13 end
14 end
15 return A, S;
16 end

Algorithm 5: ColOp(A, S, k)

/* A is an element of $SL(d, q)$ for some d and q. S is an SLP in the standard generating set. k is an integer - the number of rows that have already been killed. Return A with its first row killed and the corresponding modified SLP S. */

1 **begin**
2 **for** $j \in \{2, \ldots, d-k\}$ **do**
3 $\pi := (\overline{vu})^{j-2}$;
4 **for** $r \in \{1, \ldots, e\}$ **do**
5 **if** IsEven(r) **then**
6 $S := S(\bar{O}^{\theta})^{A_{1,j,r}}$, where $\theta := \bar{\delta}^{-\frac{r-2}{2}}\pi$;
7 AddColumn$(A, -\omega^{r-1}A_{1,j,r}, i, j)$;
8 **end**
9 **if** IsOdd(r) **then**
10 $S := (\bar{t}^{\theta})^{A_{1,j,r}}S$, where $\theta = \bar{\delta}^{-\frac{r-1}{2}}\pi$;
11 AddColumn$(A, -\omega^{r-1}A_{1,j,r}, i, j)$;
12 **end**
13 **end**
14 **end**
15 return A, S;
16 **end**

Algorithm 6: ConjByV(A, s_1, s_2)

```
/* A is an element of SL(d,q). s1 and s2 are two SLPs in the
   standard generating set. Return A conjugated by v and the
   corresponding modified SLPs. In this algorithm, A[i] refers to
   the i-th row of A.                                          */
1  begin
2      B := Transpose(A);
3      C := ZeroMatrix(GF(q), d, d);
4      C[1] := B[d];
5      {C[i+1] := -B[i] : i ∈ {1,...,d-1}};
6      A := Transpose(C);
7      C[1] := A[d];
8      {C[i+1] := -A[i] : i ∈ {1,...,d-1}};
9      if d ≠ 3 then
10         (s1, s2) := (v̄^-1 s1, s2 v̄);
11         else
12             (s1, s2) := (v̄ s1, s2 v̄^-1);
13         end
14     end
15     return C, s1, s2;
16 end
```

2.2.2 Complexity

The function GetAOne is used $d-1$ times in the algorithm. Each time it is called, it adds a multiple of one row of the input matrix A to another at most twice, at the cost of $O(d)$ field operations. Hence, in total it adds $O(d^2)$ to the complexity.

The function RowOp is also called $d-1$ times. It adds one row of the matrix A to

another at most $(d-1)e$ times at the cost of $O(d)$ field operations each time. So in total, this introduces $O(d^3e)$ into the complexity. Similarly, the function `ColOp` also introduces $O(d^3e)$.

The function `ConjByV` is called $d-1$ times and assigns $2d^2$ matrix entries, adding $O(d^3)$ to the algorithm. Hence, the complexity of the algorithm `SLWordInGen` as a whole is $O(d^3e)$.

2.3 $\mathrm{Sp}(d,q)$ in its natural representation

2.3.1 Introduction

The algorithms for solving this problem for the other classical groups all work in a similar way to the SL case, in the sense that row and column operations are used in order to kill each entry of an arbitrary matrix. We proceed by outlining the differences in each case.

In a vector space, a hyperbolic pair, with respect to a bilinear form β, is a pair of vectors e, f such that $\beta(e,f) = 1$ and $\beta(e,e) = \beta(f,f) = 0$. A hyperbolic basis for a vector space of even dimension d, is one made up from hyperbolic pairs such that the following conditions hold:

1. The basis is ordered thus: $\{e_1, f_1, \ldots, e_m, f_m\}$;
2. each pair $\{e_i, f_i\}$ satisfies $\beta(e_i, f_i) = 1$;
3. $\beta(e_i, f_j) = 0, \forall i \neq j, 1 \leq i, j \leq m$;
4. $\beta(e_i, e_j) = \beta(f_i, f_j) = 0, \forall 1 \leq i, j \leq m$.

Choose a hyperbolic basis for the vector space on which $\mathrm{Sp}(d,q)$ acts respect to such a bilinear form β.

The subgroup generated by s, u and v acts imprimitively on the hyperbolic basis vectors of the underlying space; the blocks that are permuted are the pairs $\{e_i, f_i\}$.

Once again we describe how to reduce an arbitrary element of the symplectic group to the identity using row and column operations and then explain how to conduct these operations using elements of the generating set.

2.3.2 Description of the Method

Suppose that A is an arbitrary element of $\mathrm{Sp}(d,q)$. The first step, as for the $\mathrm{SL}(d,q)$ case is to get a 1 in the (1, 1) entry of A. As in the $\mathrm{SL}(d,q)$ case, we do this by adding a suitable multiple of the second row to the first to get a 1 in the (1, 1) position. The following exceptions may occur:

1. If the (2, 1) entry of A is zero, we subtract the first row from the second by pre-multiplying A by t^s to make this entry non-zero.

2. If the (2, 1) and the (1, 1) entries of A are zero, we then permute the third to d-th rows by pre-multiplying A by various cominations of s, u and v whilst keeping the first two rows fixed in order to get a non-zero entry in the (3, 1) position of A. We then pre-multiply A by x^{v^2} to add the third row to the second, making the (2, 1) entry of A non-zero. The third to d-th rows of A are then put back to their original place.

Once A has a 1 in its (1, 1) entry, the top row of the matrix is then killed as follows:

1. Negate the first column;

2. Swap the first two columns;

3. Add a suitable multiple of the second column to the third whilst adding the same multiple of the fourth column to the first;

4. Swap the third and fourth columns;

5. Negate the fourth column;

6. Add a suitable multiple of the second column to the third whilst adding the same multiple of the fourth column to the first;

7. The top row is now (* -1 0 0 * ... *), where the asterisks represent any element of the field;

8. Use the generators u and v to permute the third to d-th columns;

9. Repeat steps 3 to 6 until the top row becomes (* -1 0 ... 0);

10. Swap the first two columns;

11. Negate the first column;

12. The top row now looks like (1 * 0 ... 0);

13. Add a suitable multiple of the first column to the second in order to kill the remaining entry.

2.3.3 Performing this Method Using the Generating Set

We now describe how to perform the above steps using the elements of the generating set.

1. 'Swap' the first two columns by post-multiplying A by s – this has the effect of also negating what was the second column.

2. x conjugated by suitable powers of δ will give an element that can be used to add the second column to the third. As described before, this element will also add a multiple of the fourth column to the first. The effects of this will be looked at in Lemma 2.3.1.

3. The third and fourth column can be swapped using usu, which has the effect of also negating what becomes the fourth column.

4. Once again, the algorithm uses x conjugated by suitable powers of δ to kill the third entry of A so that the top row is now $(* \ -1 \ 0 \ * \ \ldots *)$, as discussed above.

5. The elements u and v generate a group isomorphic to S_m and permute the columns of A without destroying the block structure $\{e_i, f_i\}$. Hence, we can use u and v to cycle the second to m-th blocks in order to work on the next block.

6. Continue in this way until the top row looks like $(* \ -1 \ 0 \ldots 0)$.

7. Apply s to swap the first two columns to get $(1 \ * \ 0 \ldots 0)^T$.

8. Use conjugates of t by δ, as in the SL case, to kill the remaining place (1, 2).

Once this last place has been killed, we find that we have also killed the second column as the following lemma shows.

Lemma 2.3.1 *Let the symplectic form of a matrix group be given by the matrix:*

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \ldots & 0 & 0 \\ -1 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & 1 & \ldots & 0 & 0 \\ 0 & 0 & -1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & 0 & 0 & \ldots & -1 & 0 \end{pmatrix}$$

If the top row of a symplectic matrix with respect to this form is $(1 \ \ 0 \ \ 0 \ \ \ldots \ \ 0)$, *then the second column of the matrix is* $(0 \ \ 1 \ \ 0 \ \ \ldots \ \ 0)$.

PROOF: Let a matrix $A \in \mathrm{Sp}(2n, q)$ have top row $(1 \ \ 0 \ \ 0 \ \ \ldots \ \ 0)$. Then A is a member of the matrix group that fixes the first basis element of the vector space on which it acts naturally. Hence, A^{-1} is of the same form and has the same top row as A. Hence $(A^{-1})^T$, the transpose of the inverse of A, has $(1 \ \ 0 \ \ 0 \ \ \ldots \ \ 0)$ as its first column.

Multiplying A by J on the right has the effect of swapping each column in pairs whilst negating what was the second column in each pair. In particular, the second column becomes negated and is swapped with the first.

$$J^{-1} = \begin{pmatrix} 0 & -1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & -1 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & -1 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

Similarly, multiplying A by J^{-1} on the left has the same effect on the rows of A. So, the second row becomes negated and is swapped with the first. Hence, the second row of A^J is $(0 \quad 1 \quad 0 \quad \dots \quad 0)$.

Now, if A is a symplectic matrix, $A^J = (A^{-1})^T$. This means that the first column of A^J is $(1 \quad 0 \quad 0 \quad \dots \quad 0)$, meaning that the second column of A must be $(0 \quad 1 \quad 0 \quad \dots \quad 0)$.

□

The above lemma shows that, once the top two rows have been killed, the first two columns are killed also. By conjugating A by u and v, the rows and columns can be permuted in order to reduce the rest of the A matrix to the identity as in the $\mathrm{SL}(d,q)$ case. If the algorithm completes and A has not been reduced to the identity, then A must not have been in $\mathrm{Sp}(d,q)$ as preserved by our form in the first place and hence, the algorithm returns 'false'.

2.3.4 Pseudo-code

As for the $\mathrm{SL}(d,q)$ case, we now summarise this algorithm in pseudo-code by breaking each part of the algorithm into smaller functions and encapsulating them into the main

function SpWordInGen. In the functions that follow, the SLPs corresponding to the standard generating set are global variables.

In addition to the simple functions that were described for the SL(d,q) case, we will also need the following:

- SwapColumns(A,i,j) swaps the columns i and j in the matrix A.

Algorithm 7: TimesByXS(A,S)

```
/* A ∈ Sp(d,q). S is an SLP. x,v and s are elements of the
   generating set defined above. Return Ax^{v^2 s} and the corresponding
   SLP. This function is not called if d = 2.                    */
1  begin
2      A := MultiplyColumn(A, −1, 1);
3      A := SwapColumns(A, 1, 2);
4      A := AddColumn(A, 1, 4, 1);
5      A := AddColumn(A, 1, 2, 3);
6      A := SwapColumns(A, 1, 2);
7      A := MultiplyColumn(A, −1, 1);
8      S := S x̄^{v^2 s};
9      return A, S;
10 end
```

Algorithm 8: TimesByS(A,S)

```
/* A ∈ Sp(d,q). S is an SLP. s is an element of the generating set
   defined above. Return As and the corresponding SLP.           */
1  begin
2      A := MultiplyColumn(A, −1, 2);
3      A := SwapColumns(A, 1, 2);
4      S := S s̄;
5      return A, S;
6  end
```

Algorithm 9: SwapRowsWithS(A, S)

```
/* A ∈ Sp(d, q). S is an SLP. s is an element of the generating set
   defined above. Return sA and the corresponding SLP.          */
1 begin
2     A := SwapRows(A, 1, 2);
3     A := MultiplyRow(A, −1, 1);
4     S := s̄S;
5     return A, S;
6 end
```

Algorithm 10: TimesBySInv(A, S)

```
/* A ∈ Sp(d, q). S is an SLP. s is an element of the generating set
   defined above. Return As^{-1} and the corresponding SLP.     */
1 begin
2     A := SwapColumns(A, 1, 2);
3     A := MultiplyColumn(A, -1, 2);
4     S := Ss̄^{-1};
5     return A, S;
6 end
```

Algorithm 11: TimesByU(A, S)

```
/* A ∈ Sp(d,q). S is an SLP. u is an element of the generating set
   defined above. Return Au and the corresponding SLP. This
   function is not called if d = 2.                                */
1 begin
2     A := SwapColumns(A, 1, 3);
3     A := SwapColumns(A, 2, 4);
4     S := Sū;
5     return A, S;
6 end
```

Algorithm 12: TimesByUSU(A, S)

```
/* A ∈ Sp(d,q). S is an SLP. u and s are elements of the generating
   set defined above. Return Ausu and the corresponding SLP. This
   swaps the third and fourth columns of A, whilst negating what
   becomes the fourth column. This function is not called if d = 2.
                                                                   */
1 begin
2     A, S := TimesByU(A, S);
3     A, S := TimesByS(A, S);
4     A, S := TimesByU(A, S);
5     return A, S;
6 end
```

Algorithm 13: TimesByUSUinv(A, S)

/* $A \in \mathrm{Sp}(d, q)$. S is an SLP. u and s are elements of the generating set defined above. Return $A(usu)^{-1}$ and the corresponding SLP. This swaps the third and fourth columns of A, whilst negating what becomes the third column. This function is not called if $d = 2$. */

1 **begin**
2 $A, S :=$ TimesByU(A, S);
3 $A, S :=$ TimesBySinv(A, S);
4 $A, S :=$ TimesByU(A, S);
5 return A, S;
6 **end**

Algorithm 14: TimesByVU(A, S)

/* $A \in \mathrm{Sp}(d, q)$. S is an SLP on the standard generating set. u and v are elements of the generating set defined above. Return Avu and the corresponding SLP. This rotates the 2 to $\frac{d}{2}$ column blocks. This function is not called if $d = 2$. */

1 **begin**
2 **for** $i \in \{2, \ldots, \frac{d}{2} - 1\}$ **do**
3 $A :=$ SwapColumns$(A, 2i - 1, d - 1)$;
4 $A :=$ SwapColumns$(A, 2i, d)$;
5 **end**
6 $S := S\bar{v}\bar{u}$;
7 return A, S;
8 **end**

Algorithm 15: TimesByX(A, S)

/* $A \in \mathrm{Sp}(d, q)$. S is an SLP on the standard generating set. v and x are elements of the generating set defined above. Return Ax^{v^2} and the corresponding SLP. This function is not called if $d = 2$. */

1 **begin**
2 $A :=$ AddColumn$(A, 1, 4, 1)$;
3 $A :=$ AddColumn$(A, 1, 2, 3)$;
4 $S := S\overline{x^{v^2}}$;
5 return A, S;
6 **end**

Algorithm 16: SwapBlock2WithBlockJ(A, S, j)

/* $A \in \mathrm{Sp}(d, q)$. S is an SLP on the standard generating set. Return the matrix A with its 3rd and 4th columns swapped with its $(2j-1)$-th and $2j$-th columns and the corresponding SLP. This function is not called if $d = 2$. */

1 **begin**
2 **if** $j \neq 2$ **then**
3 $A :=$ SwapColumns$(A, 3, 2j-1)$;
4 $A :=$ SwapColumns$(A, 4, 2j)$;
5 $S := S(u\bar{v}^{-1})^{j-2}(u\bar{v})^{j-2}(u\bar{v}^{-1})^{j-2}(u\bar{v})^{j-2}\bar{u}$;
6 **end**
7 return A, S;
8 **end**

Note that for the above function, despite line 5 having S set equal to what appears to be a long expression, our implementation is in MAGMA, which writes this longer word more sensibly as a shorter SLP. Either way the complexity will not change, but a

shorter SLP will result practically in quicker evaluations. This will also apply to similar SLP multiplications later in the thesis.

Algorithm 17: ConjByVinv(A, s_1, s_2)

```
/* A is a an element of Sp(d,q), s1 and s2 are SLPs on the standard
   generating set. Return A conjugated by v^{-1} and the
   corresponding SLPs. In this algorithm, A[i] refers to the i-th
   row of A. Note that this is not the same algorithm as ConjByV
   used in SL(d,q) as the element v is different for the symplectic
   group.                                                          */
1  begin
2      B := Transpose(A);
3      C := ZeroMatrix(GF(q), d, d);
4      {C[i] := B[i+2] : i ∈ {1,...,d-2}};
5      C[d-1] := B[1];
6      C[d] := B[2];
7      A := Transpose(C);
8      C := ZeroMatrix(GF(q), d, d);
9      {C[i] := A[i+2] : i ∈ {1,...,d-2}};
10     C[d-1] := A[1];
11     C[d] := A[2];
12     s2 := s2 v̄^{-1};
13     s1 := v̄ s1;
14     return C, s1, s2;
15 end
```

Algorithm 18: UsingT(A, s_2)

/* A is an element of $\mathrm{Sp}(d,q)$. s_2 is an SLP in the standard generating set. O is the transvection of $\mathrm{Sp}(d,q)$ with ω in its (1, 2) position. Return the matrix A, modified by having a suitable multiple of its first column added to its second to make the (1, 2) entry 0 and the corresponding modified SLP s_2. */

1 **begin**
2 $T :=$ identity SLP;
3 $\alpha := A_{1,2}$;
4 **if** $\alpha = 0$ **then**
5 return A, s_2;
6 **end**
7 **for** $r \in \{2, \ldots, e\}$ **do**
8 **if** IsEven(r) **then**
9 $T := T(\bar{O}^{\theta})^{\alpha_r}$, where $\theta = \bar{\delta}^{-\frac{r-2}{2}}$;
10 **else**
11 $T := T(\bar{t}^{\theta})^{\alpha_r}$, where $\theta = \bar{\delta}^{-\frac{r-1}{2}}$;
12 **end**
13 **end**
14 **end**
15 $A :=$ AddColumn$(A, -A_{1,2}, 1, 2)$;
16 $s_2 := s_2 T^{-1}$;
17 return A, s_2.
18 **end**

Algorithm 19: GetBetaTransvection(β)

/* β is an element of the field $\mathbb{F}$:= GF(q), where $q := p^e$ and ω is the primitive element of $\mathbb{F}$. O here is global variable calculated in Algorithm 24. This function is called only by GetAOne and therefore, β is never zero. Return the SLP representing the transvection of $\mathrm{Sp}(d,q)$ with β in its (2, 1) entry. */

1 **begin**
2 $T := \bar{t}^{-1}$;
3 **for** $r \in \{1,\ldots,e\}$ **do**
4 **if** IsEven(r) **then**
5 $T := T((O^{-1})^{\theta})^{\beta_{r-1}}$, where $\theta = \bar{\delta}^{-\frac{r-2}{2}}\bar{s}$;
6 **else**
7 $T := T((\bar{t}^{-1})^{\theta})^{\beta_{r-1}}$, where $\theta = \bar{\delta}^{-\frac{r-1}{2}}\bar{s}$;
8 **end**
9 **end**
10 **end**
11 return T;
12 **end**

Algorithm 20: IfIIsNot2(A, s_2, i)

/* A is an element of Sp(d, q) with 0 in its (1, 2) entry. s_2 is an SLP in the standard generating set. i is the earliest non-zero entry, excluding (1, 1), in the top row. Return the matrix A, modified by having its (1, 2) entry set to something non-zero and the corresponding modified SLP s_2. */

1 **begin**

2 $\beta := \frac{1-A_{1,1}}{A_{1,i}}$;

3 $T :=$ GetBetaTransvection(β);

4 $j :=$ number of the block containing i;

5 $A, s_2 :=$ SwapBlock2WithBlockJ(A, s_2, j);

6 $A, s_2 :=$ TimesByS(A, s_2);

7 $A, s_2 :=$ TimesByU(A, s_2);

8 /* we now add column 4 to 1 and column 2 to 3 so that, when we stick all the columns back again, A will have non-zero entry in the (1, 2) position */

9 **if** IsEven(i) **then**

10 $A, s_2 :=$ TimesByX(A, s_2);

11 **else**

12 $A, s_2 :=$ TimesByXS(A, s_2);

13 **end**

14 **end**

15 /* we now proceed to put all the columns back to their original positions */

16 $A, s_2 :=$ TimesByU(A, s_2);

17 $A, s_2 :=$ TimesBySInv(A, s_2);

18 $A, s_2 :=$ SwapBlock2WithBlockJ(A, s_2, j);

19 return A, s_2.

20 **end**

Algorithm 21: GetAOne(A, s_2)

/* A is an element of $\mathrm{Sp}(d,q)$. s_1 and s_2 are SLPs in the standard generating set. Return the matrix A, modified by having its (1, 1) entry set to 1 and the corresponding modified SLP s_2. */

1 **begin**

2 **if** $A_{1,1} = 1$ **then**

3 return A, s_2;

4 **end**

5 **if** $\forall i \in \{2, \ldots, d\}, A_{1,i} = 0$ **then**

6 $A :=$ AddColumn$(A, 1, 1, 2)$;

7 $s_2 := s_2\bar{t}$;

8 **end**

9 $i := \min\{2, \ldots, d : A_{i,1} \neq 0\}$;

10 /* We wish to add a suitable multiple of the second column to the first. If $A_{1,2} = 0$ then we must make it non-zero using the following function */

11 **if** $i \neq 2$ **then**

12 $A, s_2 :=$ IfIIsNot2(A, s_2, i);

13 **end**

14 $\beta := \frac{1-A_{1,1}}{A_{1,2}}$;

15 $T :=$ GetBetaTransvection(β);

16 AddColumn$(A, \beta, 2, 1)$;

17 $s_2 = s_2T$;

18 return A, s_2;

19 **end**

Algorithm 22: KillPlace(A, s_2)

```
/* A is an element of Sp(d,q).  s_2 is an SLP in the standard
   generating set.  Return the matrix A, modified by having a
   suitable multiple of its second column added to the third to make
   the (1, 3) entry 0 and the corresponding modified SLP s_2.      */
1 begin
2     α := A_{1,3};
3     for r ∈ {1,...,e} do
4         s_2 := s_2(x̄^θ)^{α_r}, where θ = v̄^2 δ̄^{r-1};
5     end
6     A := AddColumn(A, α, 4, 1);
7     A := AddColumn(A, α, 2, 3);
8     return A, s_2;
9 end
```

Algorithm 23: KillRow(A, s_2)

```
/* A ∈ Sp(d,q), with a 1 in its (1, 1) place.  Return A with its top
   row fully killed.                                                  */
1  begin
2      A, s2 := TimesByS(A, s2);
3      for i ∈ {1, ..., d/2 − 1} do
4          A, s2 := KillPlace(A, s2);
5          A, s2 := TimesByUSU(A, s2);
6          A, s2 := KillPlace(A, s2);
7          A, s2 := TimesByUSUinv(A, s2);
8          A, s2 := TimesByVU(A, s2);
9      end
10     A, s2 := TimesBySinv(A, s2);
11     A, s2 := UsingT(A, s2);
12     return A, s2;
13 end
```

Before defining the next function, we introduce some notation. Let $P = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_n x^n$ be a polynomial over one indeterminate x with coefficients in a finite field. Then P_i denotes the coefficient α_i.

Algorithm 24: SpWordInGen(G, A)

/* $G = \mathrm{Sp}(d, q)$ for some d even and q. A is an element of $\mathrm{Sp}(d, q)$. It is asserted in a pre-processing stage that A is an element of G by asserting that A has determinant 1 and that it preserves the required form. Return 'true' and an SLP for A in the standard generating set for G. */

1 **begin**

2 (s_1, s_2) :=(identity slp, identity slp);

3 /* constructing O, which is a global variable */

4 $\mathbb{K} := \langle \omega^2 \rangle_{\times,+}$;

5 $P := \omega \in \mathbb{K}$; /* P is a polynomial in ω^2 */

6 $\bar{O}$:= identity SLP;

7 **for** $i \in \{1, \ldots, e\}$ **do**

8 $\quad \bar{O} := \bar{O}(\bar{t}^{\delta-i+1})^{P_{i-1}}$;

9 **end**

10 **for** $k \in \{1, \ldots, \frac{d}{2}\}$ **do**

11 $\quad A, s_2 :=$ GetAOne(A, s_2);

12 $\quad A, s_2 :=$ KillRow(A, s_2);

13 $\quad A, s_1 :=$ SwapRowsWithS(A, s_1);

14 $\quad A, s_2 :=$ GetAOne(A, s_2);

15 $\quad A, s_2 :=$ KillRow(A, s_2);

16 $\quad A :=$ ConjByVInv(A, s_1, s_2);

17 **end**

18 assert $A = I_d$;

19 return 'true', $s_1^{-1} s_2^{-1}$;

20 **end**

2.4 SU(d, q) in its natural representation, d even

The algorithm for this case works in a similar way to the symplectic case, as the subgroup generated by s, u and v preserves a hyperbolic basis. Hence, we shall only describe the main differences between this case and the symplectic case.

The matrix representing the preserved form is:

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

As there does not necessarily exist a transvection in SU(d, q) that has a 1 in the (1, 2) entry, we cannot get the (1, 1) entry of A to be 1 in the same way. Instead we pre-multiply A by products of conjugates of x^{v^2} by powers of δ to add a suitable multiple of the third row to the first. The following exceptions may occur:

1. If the (3, 1) entry of A is zero, then we pre-multiply A by various cominations of s, u and v to permute the third to d-th rows until this is no longer the case, whilst keeping the first two rows fixed.

2. If all the entries in the first column are zero, with the exception of the (1, 1) and (2, 1) entries, then the above is not possible. Instead we first post-multiply by $x^{v^2 s}$ to add a multiple of the fourth column to the first in order to insert non-zero entries elsewhere into the first column. We then apply the process described in the above exception.

Another major difference for the unitary groups is how to proceed to kill the (1, 2) entry of the arbitrary element $A \in$ SU(d, q), having already killed the rest of the top row.

We consider odd characteristic first. As previously stated, there does not necessarily exist a transvection in $SU(d, q)$ that has a 1 in the (1, 2) entry. Hence, the (1, 2) entry of A cannot be killed in the same way as the previous two cases. We instead have the element t, which has $\alpha = \omega^{(q+1)/2}$ in its (1, 2) position.

However, as the following lemma shows, the subfield of $GF(q^2)$ generated by α, is $GF(q^2)$, if q is odd. Hence, we find the (1, 2) entry of A as a polynomial in α and use conjugates of t by powers of y to kill said entry.

Lemma 2.4.1 *If q is odd, the subfield of $F = GF(q^2)$ generated by $\alpha = \omega^{(q+1)/2}$ is F.*

PROOF: F consists of 0 and powers of ω from 1 to $q^2 - 1$. $(q^2 - 1)/(\frac{q+1}{2}) = 2(q - 1)$, so powers of α give a cyclic subgroup containing $2(q - 1)$ elements. As q is odd, $2(q - 1) > |GF(q)| = q$. Futhermore, $\alpha^2 = \omega^{q+1}$ is in $GF(q)$, whereas α is not in $GF(q)$. So the cyclic subgroup generated by α is strictly larger than $GF(q)$ and contains it, which means that it is the whole of $GF(q^2)$. □

We will now show that, at this point of the algorithm, the (1, 2) entry of A is a sum of odd powers of α. However, first we prove the following fact about trace zero elements of $GF(q^2)$ in odd characterstic. By trace zero, we mean trace zero over q. That is to say, c is of trace zero if $c + c^q = 0$.

Lemma 2.4.2 *For odd q, an element $c \in GF(q^2)$ of trace zero is a $GF(p)$-linear sum of odd powers of $\alpha = \omega^{(q+1)/2}$.*

PROOF: As c is of trace zero, $c = -c^q$. Also that $\alpha^{2(q-1)} = 1$, so $\alpha^{2q} = \alpha^2$ and hence $\alpha^q = -\alpha$, since $\alpha^q = \alpha$ would contradict the fact that $\alpha \notin GF(q)$. Hence, $\text{trace}(\alpha) = 0$. Now $c = c_0 + c_1\alpha$, where $c_0, c_1 \in GF(q)$, which means that $\text{trace}(c_0) = 0$. However, $\text{trace}(c_0) = 2c_0$, since $c_0 \in GF(q)$, which means that $c_0 = 0$ and so $c = c_1\alpha$.

Now α^2 is a primitive element of $GF(q)$, and so if $q = p^e$ then $\{\alpha^{2i} : 0 \leq i < e\}$ is a basis for $GF(q)$ over $GF(p)$. Therefore, $\{\alpha^{2i+1} : 0 \leq i < e\}$ is a basis for a complement

of GF(q) as a GF(p)-subspace of GF(q^2). □

Lemma 2.4.3 *In odd characteristic, if the top row of a matrix A preserving our chosen unitary form has the top row $(1\ x\ 0 \ldots 0)$, then x is a sum of odd powers of α.*

PROOF: The matrix representing our form is:

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \ldots & 0 & 0 \\ 1 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & 1 & \ldots & 0 & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & 0 & 0 & \ldots & 1 & 0 \end{pmatrix}$$

$\bar{A}^T$ has first column $(1\ \bar{x}\ 0 \ldots 0)$ and post-mulitplying by J moves this to the second column. $J\bar{A}^T J$ gives the second column as $(\bar{x}\ 1\ 0 \ldots 0)$.

Now, $AJ\bar{A}^T J = I$, since J is self-inverse. So we can see that the (1, 2) entry of $AJ\bar{A}^T J$ is $x + \bar{x} = 0$.

By the above lemma, x is the sum of odd powers of α and hence the proof is complete.

□

A quick calculation shows that elements of the form t^{v^i} are transvections that have odd powers of α in the (1, 2) place and hence can be used to kill said powers of α.

In even characterstic, we define α by taking the square root in F of w^{q+1}. This square root exists because every element of a finite field of characteristic 2 has a square root.

Lemma 2.4.4 *If q is even, The subfield of $F = GF(q^2)$ generated by α is $GF(q)$.*

PROOF: Firstly we note that ω^{q+1} is in GF(q) $\setminus \{0\}$ and, because q is even, it generates it multiplicatively since $\frac{q^2-1}{q+1} = q - 1 = |\text{GF}(q) \setminus \{0\}|$. However, fields of even

characterstic are perfect, so the square root of ω^{q+1} is still in GF(q). Hence α generates GF(q). □

Unlike for odd characteristic, conjugating t by powers of y will give any power of α that is required and hence, at this point of the algorithm, the (1, 2) entry of A need not be a sum of odd or even powers of α. Furthermore, in even characteristic, $\alpha = \alpha^q$, since the square root of the primitive element ω equals $\omega^{\frac{q^2}{2}}$ and so $\alpha = \sqrt{\omega^{q+1}} = \sqrt{\omega}\omega^{\frac{q}{2}} = \omega^{\frac{q^2}{2}+\frac{q}{2}}$. However, $\alpha^q = (\sqrt{\omega^{q+1}})^q = \omega^{q\frac{q+1}{2}} = \alpha$.

Lemma 2.4.5 *In even characteristic, if the top row of a matrix A preserving our chosen unitary form has the top row (1 x 0 ... 0), then x is in GF(q).*

PROOF: The matrix representing our form is:

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

$\bar{A}^T$ has first column (1 $\bar{x}$ 0 ... 0) and post-mulitplying by J moves this to the second column. $J\bar{A}^T J$ gives the second column as ($\bar{x}$ 1 0 ... 0).

Now, $AJ\bar{A}^T J = I$ since J is self-inverse. So we can see that the (1, 2) entry of $AJ\bar{A}^T J$ is $x + \bar{x} = 0$. Hence, $x = x^q$. x will be some power of the primitive element ω so let $x = \omega^i$ for some i. Now $\omega^i = \omega^{iq}$ amd so $i \equiv iq \bmod q^2-1$. Hence, $(q-1)i \equiv 0 \bmod q^2-1$ and so i is a multiple of $q+1$. As, ω^{q+1} is the primitive element of GF(q) we are done. □

2.5 SU(d, q) in its natural representation, d and q odd

2.5.1 The main algorithm

With respect to a hyperbolic basis, $H = \mathrm{SU}(d-1, q)$ lies as a subgroup of $G = \mathrm{SU}(d, q)$ in the following sense. By suitable ordering of the basis, this subgroup H can be realised in G as those $d \times d$ matrices with $(0 \ldots 0\ 1)$ as their last row and column. See Don Taylor's The Geometry of Classical Groups for details [7]. The matrix representing the form being preserved by the this group is:

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \ldots & 0 \\ 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 1 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 \end{pmatrix}.$$

Hence, for $d > 3$, we can consider using the generators of H to kill the upper $(d-1) \times (d-1)$ block of an arbitrary matrix A of G and then use the generators of G to complete the process. Hence, the algorithm needs to find the generator x of $\mathrm{SU}(d-1, q)$ as a word in the generators of $\mathrm{SU}(d, q)$. The algorithm reduces A to a matrix of dimension 3, and the method as described in Section 2.5.2 is then used to complete the process.

Now, x^{vy^t} gives a matrix with powers of ω^{q-1} in the $(1, d)$ position and these can be used to kill the last entry on the top row. This is because the subfield of $\mathbb{F}_{q^2}$ generated by ω^{q-1} is $\mathbb{F}_{q^2}$, proved below. To summarise, positions 3 to $d-1$ on the top row are killed first. Once the $(1, d)$ position has been killed, using the group elements defined at the beginning of the paragraph, the second position on the top row is killed using the same method as for the unitary groups in even dimension.

Lemma 2.5.1 *Let u, x, y and s be generators of* $\mathrm{SU}(d, q)$, *d odd. Define $s' = s(y^v)^{\frac{q^2+q}{2}}$. Let $\beta = (x^v)^{-1}s'(x^vu)^{-1}us'us'u(x^v)^{-1}$. Then, for odd characteristic, $((\beta^u s'(x^v)^{-1})^{us'u})^{-1}$*

is the generator x for $\mathrm{SU}(d-1,q)$.

PROOF: A quick calculation shows that $(y^v)^i$ is:

$$(y^v)^i = \begin{pmatrix} \omega^i & 0 & 0 & \dots & 0 & 0 \\ 0 & \omega^{-qi} & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & \omega^{(q-1)i} \end{pmatrix}$$

Let $i = \frac{q^2+q}{2}$. Then multiplying s by $(y^v)^i$ gives:

$$s' = s(y^v)^i = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & -1 \end{pmatrix}.$$

Now,

$$us'us'u = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & & & \\ \vdots & \vdots & \vdots & \vdots & & I_{d-4} & \\ 0 & 0 & 0 & 0 & & & \end{pmatrix}.$$

By calculating explicity, it can be shown that:

$$\beta = \begin{pmatrix} 1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -1 & \frac{1}{2} & 1 & \dots & 0 & -1 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & -1 & 0 & \dots & 0 & -1 \end{pmatrix}.$$

By conjugating β by u and then post-multiplying by s' we get the following:

$$\beta^u s' = \begin{pmatrix} 1 & \frac{1}{2} & 0 & -1 & \dots & 0 & 1 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & -1 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Post-multiplying by $(x^v)^{-1}$ gets rid of most of the unwanted entries:

$$\beta^u s'(x^v)^{-1} = \begin{pmatrix} 1 & 0 & 0 & -1 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Conjugating by $us'u$ gives you the inverse of the correct matrix:

$$(\beta^u s'(x^v)^{-1})^{us'u} = \begin{pmatrix} 1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Hence, by taking the inverse of the above matrix, we have the result.

□

Lemma 2.5.2 *The subfield of $F = \mathbb{F}_{q^2}$ generated by ω^{q-1} is F*

PROOF: ω^{q-1} has order $\frac{q^2-1}{q-1} = q+1 > q$. So the multiplicative group it generates is larger than $|\mathbb{F}_q|$. Hence, it must be the whole of F. □

2.5.2 SU$(3,q)$, q odd

We now proceed to describe how to reduce a matrix $A \in \mathrm{SU}(3,q)$, q odd to the identity using the standard generating set. We summarise this process as follows:

1. Let ξ be the inverse of the (1, 1) entry of A. If the (1, 1) entry is zero, we multiply A on the right by the generator s to get a non-zero entry in the (1, 1) slot. It is not possible for A to have a top row consisting of (0, 0, *), as SU$(3,q)$ with respect to the form that we are using has SU$(2,q)$ embedded as a subgroup in the top left hand corner. Another way to see that such an element could not exist in SU$(3,q)$ is that, if it did, it would map an isotropic vector to a non-isotropic vector and hence would not preserve the form. Hence, post-multiplying by s in this situation will always give a non-zero entry in the (1, 1) place. Find a zero λ of the polynomial $-\frac{1}{4}(\xi+z)(\xi^q+z)-1-z$.

2. If $\lambda \notin \mathrm{GF}(q)$ (A is written over $\mathrm{GF}(q^2)$ here), then repeatedly multiply A by the diagonal element y until we can obtain a $\lambda \in \mathrm{GF}(q)$. We prove at the end of this section that at least half of the elements of $\mathrm{GF}(q^2)$ will give rise to a $\lambda \in \mathrm{GF}(q)$ and hence the algorithm remains polynomial.

3. Let $\theta = \xi + \lambda$ and create the following matrix as an SLP in the standard generators:

$$\begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2}N(\theta) & 1 & -\theta^q \\ \theta & 0 & 1 \end{pmatrix},$$

where $N(\theta) = \theta^{q+1}$ denotes the norm of θ.

4. Multiply this matrix on the left by the generator x to give a matrix with ξ as its (1, 1) entry.

5. Now kill the (1, 3) and (3, 1) entries of the above matrix. This automatically kills the (2, 3) and (3, 2) entries, as A preserves a unitary form.

6. Add a suitable multiple of row 1 to row 2 to kill the (2, 1) entry and add a suitable multiple of column 1 to column 2 to kill the (1, 2) entry.

7. We now have a diagonal matrix with ξ as its (1, 1) entry. Multiplying A by this matrix makes the (1, 1) entry of A equal to 1.

8. A is now reduced to the identity using techniques outlined in the previous section.

We now go into the detail of the above points. The top row of the generator x is (1, $-\frac{1}{2}$, 1) and we wish to create a matrix with first column $(1, -\frac{1}{2}N(\theta), \theta)$. Multiplying these two matrices together should give ξ in the (1, 1) position. Hence, we need to solve the following equation: $1 + \frac{1}{4}N(\theta) + \theta = \xi$. Suppose that $\theta = \xi + \lambda$. Then we can rewrite this equation as $1 + \frac{1}{4}N(\xi + \lambda) + \xi + \lambda = \xi$, which we can rearrange to give $-\frac{1}{4}N(\xi + \lambda) = \lambda + 1$. As the norm of any element of $\mathrm{GF}(q^2)$ is in $\mathrm{GF}(q)$, λ must be in $\mathrm{GF}(q)$. Hence, we wish to solve the equation $-\frac{1}{4}N(\xi + \lambda) - \lambda - 1 = 0$, where $\lambda \in$

$\mathrm{GF}(q)$. If no solution exists, the original matrix A will be multiplied by the generator y until a solution in $\mathrm{GF}(q)$ can be obtained. $N(\xi+\lambda)$ can be rewritten as $(\xi+\lambda)^q(\xi+\lambda)$, which equals $(\xi^q+\lambda)(\xi+\lambda)$ since $\lambda \in \mathrm{GF}(q)$. Hence, we wish to solve the equation $-\frac{1}{4}(\xi+z)(\xi^q+z)-1-z$.

Given a solution λ, we now discuss how to construct the required matrix:

$$\begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2}N(\theta) & 1 & -\theta^q \\ \theta & 0 & 1 \end{pmatrix} .$$

Conjugating the transpose of the generator x by powers of y will give any power of ω^{q-2} in the (3, 1) place. As ω^{q-2} generates the whole of $\mathrm{GF}(q^2)$, we can get a matrix of the form:

$$\begin{pmatrix} 1 & 0 & 0 \\ \zeta & 1 & -\theta^q \\ \theta & 0 & 1 \end{pmatrix} ,$$

for any $\theta \in \mathrm{GF}(q^2)$. This process does not use discrete logs as θ can be written as a $\mathrm{GF}(q)$-linear sum of powers of ω^{q-2}. The entry ζ will be an element of the form $-\frac{1}{2}N(\theta)+c$, where c is an element of trace 0 (proved below). Hence, c can be removed using the generator t^{-1} conjugated by powers of y and we get a matrix of the required form.

Pre-multiplying the above matrix by x, gives a 3×3 matrix with ξ in the (1, 1) slot. The (1, 3), (3, 1), (2, 3) and (3, 2) entries are then killed by adding suitable multiples of the first column / row to the third using conjugates of x and t by y. This matrix is then pre-multiplied by A to give a 1 in in the (1, 1) slot of A. The algorithm subsequently reduces A to the identity using the same techniques as outlined in the previous section.

Lemma 2.5.3 *The element ζ in the above matrix, is of the form $-\frac{1}{2}N(\theta)+c$, where c is of trace 0.*

PROOF: Call the matrix above M. The matrix preserves the unitary form J and so satisfies the equation $\bar{M}^T J M = J$. That is to say, the following equation holds:

$$\begin{pmatrix} 1 & \zeta^q & \theta^q \\ 0 & 1 & 0 \\ 0 & -\theta & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ \zeta & 1 & -\theta^q \\ \theta & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} \zeta^q + \zeta + N(\theta) & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

So $\zeta^q + \zeta + N(\theta) = 0$. Let $F = \mathrm{GF}(q)$ and $K = \mathrm{GF}(q^2)$. Then $[K : F] = 2$ and so the trace of ζ is $\zeta^q + \zeta$ giving us $\mathrm{trace}(\zeta) + N(\theta) = 0$. Since $N(\theta) \in F$, $\mathrm{trace}(N(\theta)) = 2N(\theta)$ and hence $\mathrm{trace}(\zeta) + \mathrm{trace}(\frac{1}{2}N(\theta)) = 0$. So we have $\mathrm{trace}(\zeta + \frac{1}{2}N(\theta)) = 0$, which implies that $\zeta = c - \frac{1}{2}N(\theta)$, with c having trace 0. □

By Lemma 2.4.2 above, as c is an element of trace 0, it is a $\mathrm{GF}(q)$-linear sum of odd powers of $\alpha = \omega^{(q+1)/2}$, and hence can be killed using the generator t. This completes step 3 above. To complete this section, we give the following lemma.

Lemma 2.5.4 *More than half of the elements ξ of $GF(q^2)$ will give rise to an equation $f(z) = -\frac{1}{4}(\xi + z)(\xi^q + z) - 1 - z = 0$ with a solution in $GF(q)$, when q is odd. Hence, the algorithm remains polynomial for odd q.*

PROOF: The above equation can be rewritten as $z^2 + z(\mathrm{trace}(\xi) + 4) + (N(\xi) + 4)$, where $N(\xi)$ denotes the norm of ξ. If $\xi \in \mathrm{GF}(q^2) \setminus \mathrm{GF}(q)$ then there are $q^2 - q$ choices for ξ, and so $\frac{1}{2}(q-1)q$ choices for the pair $(N(\xi), \mathrm{trace}(\xi))$, since ξ and ξ^q give rise to the same pair.

We require to prove that if $z^2 + bz + c$ is irreducible over $\mathrm{GF}(q)$, then the probability that $z^2 + (b+4)z + c + 4$ is reducible is more than a half. The first equation is irreducible if and only if $b^2 - 4c$ is a non-square. The second equation is reducible if and only if $b^2 + 8b - 4c$ is a square. Consider the following simultaneous equations in b and c, in $\mathrm{GF}(q)$:

$$b^2 - 4c = u \tag{2.1}$$

$$b^2 + 8b - 4c = v \tag{2.2}$$

Given (2.1), there is exactly one c for a given b. Now (2.1) and (2.2) together determine b exactly and hence (2.1) and (2.2) have a unique solution for any u and v. Conversely, b and c determine u and v. Thus there is a bijection between ordered pairs (u, v) and ordered pairs (b, c) connected by these equations. We are given b and c as the trace and norm of ξ respectively and we want to know the probability that v is a square. Suppose that u is not a square, but v is. Then there are $\frac{1}{4}(q^2 - 1)$ pairs (u, v). This breaks down as a choice of $\frac{1}{2}(q-1)$ for u, the number of non-squares in GF(q), and a choice of $\frac{1}{2}(q+1)$ for v, the number of squares in GF(q). Hence, there are $\frac{1}{4}(q^2 - 1)$ such pairs (b, c). All these pairs give rise to two possible values for ξ, since u is not a square. So $\frac{1}{2}(q^2 - 1)$ values of ξ define (u, v) with u a non-square and v a square.

These are the ξ not in GF(q). If ξ is in GF(q) then, $v = (2\xi+4)^2 - 4(\xi^2+4) = 4^2(\xi)$. Hence, v is a square whenever ξ is and ξ is a square for $\frac{1}{2}(q+1)$ of the elements of GF(q). So the total number of possible ξ is $\frac{1}{2}(q^2 - 1) + \frac{1}{2}(q+1) = \frac{1}{2}q(q+1)$.

Now, we look at the proportion of elements that this covers: $\frac{1}{2}q(q+1)q^{-2} = \frac{1}{2} + \frac{1}{2q}$, which is greater than a half. □

2.6 SU(d, q) in its natural representation, d odd and q even

2.6.1 The generating set

We now consider even characteristic. In this subsection, we shall exhibit the generating set for SU(d, q) and show how SU$(d-1, q)$ embeds into it. The algorithm will then work as for odd characteristic and so we do not discuss this here. The generating set is the following:

Group	s	t	δ	u	v
$\mathrm{SU}(2n+1,q)$	$\begin{pmatrix} 0 & \alpha \\ \alpha^{-q} & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & \alpha \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega^{q+1} & 0 \\ 0 & \omega^{-(q+1)} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	$(e_1, e_2, \ldots, e_n)(f_1, f_2, \ldots, f_n)$
	x	y			
	$\begin{pmatrix} 1 & \phi & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 & 0 \\ 0 & \omega^{-q} & 0 \\ 0 & 0 & \omega^{q-1} \end{pmatrix}$			

Table 2.3: Standard generators for $\mathrm{SU}(d,q)$ for d odd and q even

In the above table, α represents the field element $\omega^{(q^2+q)/2}$, which is a square root of ω^{q+1}. Consider the polynomial $z^q + z + 1 = 0$ over GF(q^2). Then ϕ in the above table is a root of this equation chosen so that it is the same every time the algorithm is run. This is done by finding a solution set to the equation as powers of the primitive element and choosing ϕ to be the solution with the smallest power. For the group SU(3, 2), we also require the following extra generator:

$$\hat{x} = \begin{pmatrix} 1 & \omega & \omega \\ 0 & 1 & 0 \\ 0 & \omega^2 & 1 \end{pmatrix}.$$

2.6.2 The embedding of SU$(d-1,q)$ into SU(d,q)

We now show how to embed the generator $x \in \mathrm{SU}(2d,q)$ into $\mathrm{SU}(2d+1,q)$.

Lemma 2.6.1 *Let u, x, y and s be generators of* SU$(2d+1,q)$. *Define $s' = s(y^v)^{\frac{q^2+q}{2}}$. Let $\beta = (x^v)^{-1}s'(x^vu)^{-1}us'us'u(x^v)^{-1}$. Then, for even characteristic, $(x^{vs'v})^2(x^v)^2((\beta^u s'(x^v)^{-1})^{us'u})^{-1}$ is the generator x for* SU$(2d,q)$.

PROOF: A quick calculation shows that $(y^v)^i$ is:

$$(y^v)^i = \begin{pmatrix} \omega^i & 0 & 0 & \dots & 0 & 0 \\ 0 & \omega^{-qi} & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & \omega^{(q-1)i} \end{pmatrix}$$

Let $i = \frac{q^2+q}{2}$. Then multiplying s by $(y^v)^i$ gives:

$$(y^v)^i = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

Now,

$$us'us'u = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & & & \\ \vdots & \vdots & \vdots & \vdots & & I_{d-4} & \\ 0 & 0 & 0 & 0 & & & \end{pmatrix}$$

By calculating explicity, it can be shown that:

$$\beta = \begin{pmatrix} 1 & 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \phi^{-1} & 1 & \dots & 0 & 1 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 1 \end{pmatrix}$$

then:

$$((\beta^u s'(x^v)^{-1})^{us'u})^{-1} = \begin{pmatrix} 1 & 1 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

Pre-multiplying by $(x^{vs'v})^2(x^v)^2$ gives the result.

□

2.6.3 SU$(3,q)$, q even

This works as for odd characteristic, except that the polynomial that needs to be solved is $f(x) = \phi^2(\xi + x)^{q+1} + 1 + x$. This is because we are multiplying a matrix that has top row $(1, \phi, 1)$ with one that has first column $(1, \phi N(\theta), \theta)$.

We now prove the equivalent of Lemma 2.5.3 for even characteristic.

Lemma 2.6.2 *The element ζ (as in Lemma 2.5.3) is of the form $\phi N(\theta) + c$, where c is of trace 0.*

PROOF: Call the matrix M as before. The matrix preserves a unitary form J and so satisfies the same equation as for odd characteristic: $\bar{M}^T J M = J$, and hence $\zeta^q + \zeta + N(\theta) = 0$ as before. We again have $\text{trace}(\zeta) + N(\theta) = 0$. Consider $\text{trace}(\phi N(\theta))$. Now, this is $\phi^q N(\theta)^q + \phi N(\theta) = (\phi^q + \phi)N(\theta) = \text{trace}(\phi)N(\theta)$, since $N(\theta) \in \text{GF}(q)$. As the definition of ϕ means that it has trace 1, we have shown that $\text{trace}(\phi N(\theta)) = N(\theta)$. Hence, $0 = \text{trace}(\zeta) + N(\theta) = \text{trace}(\zeta) + \text{trace}(\phi N(\theta)) = \text{trace}(\zeta + \phi N(\theta))$, which implies that $\zeta = c - \phi N(\theta)$, with c having trace 0. □

Note that as $c = c^q$, when c is of trace 0, c is in GF(q). Furthermore, as it is possible to obtain all powers of α using products of conjugates of t by y, we do not need to prove an equivalent of Lemma 2.4.2.

Once again, we conclude the section with an equivalent lemma to the odd characteristic case.

Lemma 2.6.3 *Let ξ be in GF(q^2) and $f(z) = \phi^2(\xi + z)^{q+1} + 1 + z$. Then $f(z) = 0$ has a solution in GF(q) for more than half the elements of GF(q^2). Hence, the algorithm is polynomial in even characteristic.*

PROOF: The above equation can be rewritten as $z^2+z(\text{trace}(\xi)+\phi^{-2})+(N(\xi)+\phi^{-2})$, where $N(\xi)$ denotes the norm of ξ. If $\xi \in \text{GF}(q^2) \setminus \text{GF}(q)$ then there are $q^2 - q$ choices for ξ, and so $\frac{1}{2}(q-1)q$ choices for the pair $(N(\xi), \text{trace}(\xi))$, since ξ and ξ^q give rise to the same pair.

This equation is always solvable since every element of a field of characteristic 2 is a square. These are the ξ not in GF(q). If z is in GF(q) then the original equation can always be solved, so the total number of possible ξ is $\frac{1}{2}q(q-1) + q = \frac{1}{2}q(q+1)$.

As in odd characteristic, the proportion of elements that this covers is $\frac{1}{2}q(q+1)q^{-2} = \frac{1}{2} + \frac{1}{2q}$, which is greater than a half. □

2.7 $\Omega^+(d, q)$ in its natural representation, d even

The algorithm for this case works almost exactly the same as the symplectic group case. There are, however, two main differences. Firstly, as the matrices are preserving an orthogonal form, we find that once the third to d-th entries on the top row of an arbitrary matrix A have been killed, the second entry is automatically killed as the lemma below shows. The second is how we kill the last 4×4 block of a matrix, having already killed the remaining entries. This is discussed in Section 2.7.1.

Lemma 2.7.1 *Suppose that $A \in \Omega^+(d,q)$ is an orthogonal matrix with respect to the following form:*

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

Suppose further than the top row of A is $(1\ a\ 0 \dots 0)$. Then $a = 0$.

PROOF: As A is an orthogonal matrix with respect to a hyperbolic basis, the following equation holds: $AJA^T = J$. Note that J is self-inverse.

As stated, the top row of A is $v = (1\ a\ 0 \dots 0)$. Then transposing A gets v as the first column. Postmultiplying by J gets v as the second column and subsequently premultiplying by J gets the second column as $(a\ 1\ 0 \dots 0)$. This is the second column of JA^TJ. Consider premultiplying this by A to give the identity matrix. This gives the second entry in the top row as $2a = 0$. So in odd characteristic, $a = 0$ and we are done.

For even characteristic, if A takes e_1 to $e_1 + af_1$ then $Q(e_1 + af_1) = 0$, where Q is a quadratic form on an ordered basis of hyperbolic pairs $\{e_i, f_i\}$. The matrix representing this quadratic form is:

$$J = \begin{pmatrix} 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix}.$$

The standard generators for characteristic 2 are the same as that for odd dimension. So $Q(e_1) + a^2Q(f_1) + ae_1.f_1 = 0$. But $Q(e_1) = Q(f_1) = 0$, so $a = 0$.

□

2.7.1 Killing the final 4 × 4 block of a matrix

As in SL, the symplectic and the unitary groups, the algorithm proceeds by killing rows and columns of an arbitrary element A of $\Omega^+(d,q)$. However, unlike for these other cases, we are not able to reduce the matrix to the identity by using row and column operations alone. This is because, if we kill all bar the last 2×2 block, we are left with a diagonal element that can only be reduced to the identity by using discrete logarithms, which will add an exponential complexity to the algorithm. Instead, we do the following:

1. Kill everything except the lower 4×4 block of the matrix A. Extract this matrix and call it A'.

2. Apply the following change of basis to A' as defined in the paper of Leedham-Green and O'Brien [4]:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

3. As $\Omega^+(4,q)$ is isomorphic to the central product of two copies of $\mathrm{SL}(2,q)$, we can recognize A' as the tensor product (disregarding scalars) of two elements h_1 and h_2 of $\mathrm{SL}(2,q)$.

4. Write h_1 and h_2 as SLPs in the standard generators of $\mathrm{SL}(2,q)$ using `SLWordInGen`.

5. Eight of the standard generators (all bar u and v) of $\Omega^+(d,q)$ are formed by two sets Q_1 and Q_2: Q_1 is the tensor product of I_2 with the standard generators of $\mathrm{SL}(2,q)$ and Q_2 is the tensor product of $\mathrm{SL}(2,q)$ with the standard generators of I_2. Hence, we can consider the SLPs that we have for h_1 and h_2 as SLPs in Q_1

and Q_2. Therefore, we have h_1 and h_2 as words in the generating set of $\Omega^+(d,q)$ and so we have a word for A'. The algorithm is therefore complete.

2.7.2 $\mathrm{SO}^+(d,q)$

In their paper, Leedham-Green and O'Brien provide a generator that, when added to the generators for $\Omega^+(d,q)$, forms a generating set for $\mathrm{SO}^+(d,q)$. The generator is:

$$\sigma = \begin{pmatrix} \omega^b & 0 & \dots & 0 \\ 0 & \omega^{-b} & \dots & 0 \\ \vdots & \vdots & I_{d-2} & \\ 0 & 0 & & \end{pmatrix},$$

where ω is the primitive element of $\mathrm{GF}(q)$ and b is determined by $q-1 = 2^a b$, where b is odd.

We outline an algorithm to solve the word problem for $\mathrm{SO}^+(d,q)$ as follows. Choose $g \in G = \mathrm{SO}^+(d,q)$. We then calculate the spinor norm of g. This will be 0 if $g \in \Omega^+(d,q)$ and 1 if g is in the other coset of $\mathrm{SO}^+(d,q)$. If $g \in \Omega^+(d,q)$, then the algorithm returns the required word using `OmegaPlusWordInGen`. If $g \notin \Omega^+(d,q)$, then the $\Omega^+(d,q)$ code is performed on $g\sigma$. The output will be a word w and the word for g will therefore be $w\bar{\sigma}^{-1}$, where $\bar{\sigma}$ is the element of the SLP representing σ.

2.8 $\Omega^-(d,q)$ in its natural representation, d even and q odd

2.8.1 Introduction

This case is markedly different from all the other cases. With respect to a hyperbolic basis with a carefully chosen ordering, $H = \Omega^+(2d-2,q)$ can be realised as a subgroup of $G = \Omega^-(2d,q)$ as those matrices consisting of the form:

$$\begin{pmatrix} & & & 0 & 0 \\ & H & & \vdots & \vdots \\ & & & 0 & 0 \\ 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

See Don Taylor's The Geometry of Classical Groups for details [7]. So, in a similar way to the unitary case in odd dimension, we first wish to find the generators of $\Omega^+(2d-2,q)$ in terms of the generators of $\Omega^-(2d,q)$, in order to kill the third to $(2d-2)$-nd entry of the top to $(2d-4)$-th row of an element of $\Omega^-(2d,q)$.

Lemma 2.8.1 *Let* $B(h) = (h^{v^2})^{-1}(h^v)^{\frac{q-1}{2}}h^{v^2}$ *and* $B'(h) = ((hs)^{v^2})^{-1}(h^v)^{\frac{q-1}{2}}(hs)^{v^2}$, *where* v *and* s *are generators for* $\Omega^-(2d,q)$ *and* h *is some other generator. For* $t \in \Omega^-(2d,q)$, *let* a *(respectively* a'*) be the (1, 1) entry of* $B(t)$ *(respectively* $B'(t)$*),* b *(*b'*) the (1, d-1) entry and* c *(*c'*) the (1, 2) entry. Let* $F = GF(q)$, *let* n *be a solution in* F *to the quadratic* $ax^2 + 2bx + c = 0$ *and let* m *be a solution in* F *to* $a'x^2 + 2b'x + c' = 0$. *Then for odd characteristic:*

- $t \in \Omega^+(2d-2,q)$ *is formed by* $(t^v)^n B(t) \in \Omega^-(2d,q)$;
- $r \in \Omega^+(2d-2,q)$ *is formed by* $(r^v)^n B(r) \in \Omega^-(2d,q)$;
- $t' \in \Omega^+(2d-2,q)$ *is formed by* $(t^v)^m B'(t) \in \Omega^-(2d,q)$;
- $r' \in \Omega^+(2d-2,q)$ *is formed by* $(r^v)^m B'(r) \in \Omega^-(2d,q)$;

PROOF: Consider $(t^{v^2})^{-1}(h^v)^j t^{v^2}$. A simple calculation shows that this gives a matrix with the top 4×4 block looking like this:

$$\begin{pmatrix} 1 & * & 0 & 2j \\ 0 & 1 & 0 & 0 \\ 0 & -2j & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

It is easy to see that setting $j = \frac{q-1}{2}$ will give the matrix

$$\begin{pmatrix} 1 & * & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

We now need to work out how to set the remaining entry, denoted *, to 0. The full matrix of $B(t)$ has the form:

$$\begin{pmatrix} 1 & b & 0 & -1 & 0 & \dots & 0 & c & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & a & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

Consider how t^v acts on $B(t)$ on the left.

$$t^v = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 1 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 2 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

Let the top row be v_1, the second row be v_2 and the $(d-1)$- th row be v_3. Premultiplying $B(t)$ by t^v sends $v_1 \mapsto v_1 + v_2 + v_3$, $v_2 \mapsto v_2$ and $v_3 \mapsto v_3 + 2v_2$. Therefore, premultiplying by $(t^v)^n$ sends $v_2 \mapsto v_2$ and $v_3 \mapsto v_3 + 2nv_2$.

We show that we have $v_1 \mapsto v_1 + n^2 v_2 + n v_3$ by induction, the case where $n = 1$ being trivial. Suppose that $(t^v)^{i-1}$ maps $v_1 \mapsto v_1 + (i-1)^2 v_2 + (i-1)v_3$, then premultiplying by t^v gives $v_1 \mapsto (v_1 + (i-1)^2 v_2 + (i-1)v_3) + (v_2) + (v_3 + 2(i-1)v_2) = v_1 + i^2 v_2 + i v_3$. So the induction follows. Setting $v_1 = b$, $v_2 = 1$ and $v_3 = a$ gives as a quadratic equation that we can solve for n to get the power of t^v needed to kill the (1, 2) entry and so we're done.

The other three equations can be shown to hold by a similar method. □

2.8.2 Description of the Method

We now describe how this algorithm works for $\Omega^-(2d, q)$.

1. Add a suitable multiple of the fourth row to the first to get the (1, 1) entry of the matrix A to be 1, using the same techniques for the Ω^+ case.

2. Using the generators for $\Omega^+(2d-2, q)$, kill the third to $(2d-2)$-nd entry in the top row. The top row now looks like this: $(1 * 0 \ldots 0 * *)$, where the asterisks represent entries in the field GF(q).

3. In a similar way, use the generators for $\Omega^+(2d-2, q)$, to kill the third to $(2d-2)$-nd entry in the first column.

4. We now wish to kill the $(d-1)$-th and dth slots in the top row, which will automatically kill the second slot. We will do this using conjugates of t^v by powers of δ. As t and δ are gained from the tensor product of elements of $\mathrm{SL}(d, q^2)$ and performing a basis change, we wish to consider how to kill these remaining two entries in the top row of A by considering the equivalent problem in $\mathrm{SL}(2, q^2)$. The equivalent problem in $\mathrm{SL}(2, q^2)$ is having a matrix of the following form:

$$\hat{A} = \begin{pmatrix} 1 & a \\ * & * \end{pmatrix},$$

and wishing to kill the entry a by adding a multiple of the first column to the second. Let $\hat{t}$ and $\hat{\delta}$ be the equivalent matrices for t and δ in $\mathrm{SL}(2, q^2)$. That is to say $\hat{t} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $\hat{\delta} = \begin{pmatrix} \gamma & 0 \\ 0 & \gamma^{-1} \end{pmatrix}$, where γ is the primitive element of $\mathrm{GF}(q^2)$. From the description of the $\mathrm{SL}(d, q)$ above, we can see that this process is performed by finding conjugates of t by powers of δ so that we have a set of matrices of the form $\hat{K} = \{\begin{pmatrix} 1 & \gamma^i \\ 0 & 1 \end{pmatrix} : i \in \{0, \ldots, e-1\}\}$, where e is the degree of the field. Then these matrices are used to add the correct multiple of column 1 to column 2 to kill a. Hence, we create a set of matrices K in $\Omega^-(2d, q)$, created from conjugates of t^v by powers of δ, which are the image of the matrices $\hat{K}$ in $SL(2, q^2)$.

5. The matrices in K are of the following form:

$$\begin{pmatrix} 1 & 1 & 0 & \ldots & 0 & a & b \\ 0 & 1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 & 0 \\ 0 & * & 0 & \ldots & 0 & 1 & 0 \\ 0 & * & 0 & \ldots & 0 & 0 & 1 \end{pmatrix}.$$

Consider the vector $(a, b) \in (\mathbb{F}_q)^2$. This can be considered as a vector over $(\mathbb{F}_p)^{2e}$, where e is the degree p is the characterstic of the field respectively. As there is one matrix in K for each one in $\hat{K}$, the matrices in K can be considered as a basis in this way for $(\mathbb{F}_p)^{2e}$. Similarly, we can consider the $(d-1)$-th and d-th entries of the top row of A to be a vector in $(\mathbb{F}_p)^{2e}$. Call this vector u. Write u in terms of the basis afforded by K and call this new vector u.

6. Suppose $u = (u_1, \ldots, u_{2e})$ and let K_i be the matrix in K corresponding to the matrix $\begin{pmatrix} 1 & \gamma^{i-1} \\ 0 & 1 \end{pmatrix}$ in $\hat{K}$. Then we proceed by killing the remaining entries in the top row of A by setting $A := AK_1^{-u_1}K_2^{-u_2}\ldots K_{2e}^{-u_{2e}}$.

7. This process is then dualised so that the remaining entries in the first column are killed. As A preserves an orthogonal form, we have now killed the first two rows and columns.

8. We then proceed working down the matrix recursively until we are left with a 4 $\times$ 4 block in the bottom right hand corner of A.

9. As $\Omega^-(4,q)$ is isomorphic to $\mathrm{PSL}(2,q^2)$, the module arising as the tensor product of the natural module for $\mathrm{SL}(2,q^2)$ tensored with its Frobenius q-twist, we proceed by recognizing the remaining 4 $\times$ 4 block as an element of $\mathrm{SL}(2,q^2)$ to kill the remaining block off. Call this block A'. We wish to find the equivalent matrix in $\mathrm{SL}(d,q^2)$. As Leedham-Green and O'Brien in [4] created the generators for $\Omega^-(4,q)$ by tensoring generators of $\mathrm{SL}(2,q^2)$ and applying a change of basis, this process is reversed to recognize A' as an element of $\mathrm{SL}(2,q^2)$.

10. The image of A' in $\mathrm{SL}(2,q^2)$ is then written as a word in the image of the generators in $\mathrm{SL}(2,q^2)$ and, as this word is written as an SLP, it is subsequently considered as a word in the generating set of $\Omega^-(4,q)$ and hence we are done due to the lemma below.

Lemma 2.8.2 *If the element A that we are testing for membership is in $\Omega^-(d,q)$, then the word for A', obtained by the method outlined in point 10 above, will not evaluate to $-A'$ on the standard generators for $\Omega^-(d,q)$.*

PROOF: Even though $\Omega^-(4,q)$ is isomorphic to $\mathrm{PSL}(2,q^2)$, the word for A', when evaluated on the generating set for $\Omega^-(4,q)$, can only evaluate to A' and not $-A'$. Suppose, for contradiction, that the word for A' garnered from the aforementioned method evaluated to $-A'$. This implies that $-A' \in \Omega^-(4,q)$. Then, given that also $A' \in \Omega^-(4,q)$, this means that $-A'A'^{-1} = -I_4 \notin \Omega^-(4,q)$. Hence, we have a contradiction. □

2.8.3 $\mathrm{SO}^-(d,q)$

The method for solving the problem for $\mathrm{SO}^-(d,q)$ is the same as that for $\mathrm{SO}^+(d,q)$. The only exception is that the extra generator needed, as defined in the paper by Leedham-Green and O'Brien is:

$$\sigma = \begin{pmatrix} \lambda I_2 & 0 \\ 0 & -\lambda I_2 \end{pmatrix},$$

where $\lambda = (-1)^{(q-1)/2}$.

2.9 $\Omega^-(d,q)$ in its natural representation, d and q even

2.9.1 Forming the Generating Set

We now consider even characteristic. In this subsection, we shall exhibit the generating set for the group and show how $\Omega^+(d-2,q)$ embeds into it. The algorithm will then work as for odd characteristic and so we do not discuss this here. The generating set for this case is created in a similar way to how it is created for odd characteristic; that is by considering the isomorphism between $\Omega(4,q)$ and $\mathrm{PSL}(2,q^2)$. This isomorphism arises as follows. Take the natural module U for $\mathrm{SL}(2,q^2)$, and let W be U twisted by the automorphism of $\mathrm{GF}(q^2)$ given by $a \mapsto a^q$. Then $U \otimes W$ gives rise to a representation of $\mathrm{PSL}(2,q^2)$ over $\mathrm{GF}(q^2)$. If (a_1,b_1) is a basis for U, and (a_2,b_2) is a basis for W, then the resulting representation of $\mathrm{PSL}(2,q^2)$ on $U \otimes W$ with respect to the ordered basis $(a_1 \otimes a_2, a_1 \otimes b_2, b_1 \otimes a_2, b_1 \otimes b_2)$ preserves the symmetric non-degenerate bilinear form:

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Now let γ be a primitive element of $\mathrm{GF}(q^2)$. Conjugating by the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \gamma & 1 & 0 \\ 0 & \gamma^q & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

transforms the above image of $\mathrm{PSL}(2,q^2)$ into a subgroup of $\mathrm{SL}(4,q)$. Interchanging the second and fourth basis vectors now transforms this image into a group that preserves the form:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \gamma+\gamma^q \\ 0 & 0 & \gamma+\gamma^q & 0 \end{pmatrix},$$

and hence into our chosen copy of $\Omega^-(4,q)$. It is straightforward to check that the given generators s, t, δ are the images of the matrices

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \gamma & 0 \\ 0 & \gamma^{-1} \end{pmatrix},$$

given in table 2.9.1 and hence generate $\Omega^-(4,q)$. It follows, as for odd characterstic, that these generators, together with u and v, generate $\Omega^-(2n,q)$.

In the below table, the A, B and C in δ represent the following values:

$$\begin{aligned} A &= \gamma^{-1}+\gamma^{-q} \\ B &= \gamma+\gamma^q \\ C &= \gamma^{-q+1}+\gamma^{q-1}-1. \end{aligned}$$

The quadratic form of $\Omega^-(4,q)$ is the following:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \gamma+\gamma^q \\ 0 & 0 & 0 & \omega \end{pmatrix},$$

where ω is the primitive element of GF(q). The matrix of a quadratic form is upper triangular, hence the (i,j)-th position of the matrix is 0 when $i > j$. When $i < j$, the form is the same as the bilinear form. This just leaves the diagonal entries. As the first two basis vectors are isotropic, the (1, 1) and (2, 2) entries are 0. The (3, 3) entry can be chosen to be 1 by Lemma 11.1 in Don Taylor's book [7]. Consider how the generator δ in the table below acts on the last basis vector b_2. This gives $b_2\delta = (0, 0, B, C)$, where B and C are as above.

Let the (4, 4) entry of the quadratic form Q be x. As $Q(b_2) = Q(b_2\delta)$, we have that $x = B^2 + (\gamma+\gamma^q)BC + xC^2$, by the formula $Q(a_ie_i) = \sum_{i\leq j} c_{ij}a_ia_j$, where the c_{ij} are the entries in the form and a_i is the i-th entry of the vector. As $B = \gamma+\gamma^q$, this can be rewritten as $x = B^2 + B^2C + xC^2$. Then, $x(1+C^2) = B^2(1+C)$ and the left hand side can be rewritten as $x(1+C)^2$. Hence, $x(1+C) = B^2$. But $(1+C) = AB$ and so $xA = B$. Let $x = \gamma^i$, for some i. Then the above equation can be rewritten as $\gamma^{i-1} + \gamma^{-q+i} = \gamma+\gamma^q$ and so $i = q+1$, meaning $x = \gamma^{i+1} = \omega$.

Group	t	r	δ	u	v
$\Omega^-(2n,q)$	$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \gamma+\gamma^q & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ \gamma+\gamma^q & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} \omega & 0 & 0 & 0 \\ 0 & \omega^{-1} & 0 & 0 \\ 0 & 0 & 1 & A \\ 0 & 0 & B & C \end{pmatrix}$	$(e_1,e_2)(f_1,f_2)$	$(e_1,\ldots,e_{n-1})(f_1,\ldots,f_{n-1})$

Table 2.4: Standard generators for orthogonal groups in characterstic 2

2.9.2 The Embedding of $\Omega^+(d-2,q)$ into $\Omega^-(d,q)$

Lemma 2.9.1 *Let $B(h) = (h^{v^2})^{-1}((h^\delta)^v)h^{v^2}$, where v and δ are generators for $\Omega^-(2d,q)$ as they appear in the above table, h is some other generator and ω is the primitive element of the ground field. Let α_i be the coefficients of the (4, 1) entry of $B(h)$ written as a polynomial in the primitive element and let $\prod(h) = \prod_{i=1}^{e}((h^v B(t))^{\delta^{i-1}})^{\alpha_i}$. Then for even characteristic:*

- $t \in \Omega^+(2d-2,q)$ *is formed by* $t^v B(t)\prod(t) \in \Omega^-(2d,q)$;
- $r \in \Omega^+(2d-2,q)$ *is formed by* $r^v B(r)\prod(r) \in \Omega^-(2d,q)$;
- $t' \in \Omega^+(2d-2,q)$ *is formed by* $(t^v B(t)\prod(t))^s \in \Omega^-(2d,q)$;
- $r' \in \Omega^+(2d-2,q)$ *is formed by* $(r^v B(r)\prod(r))^s \in \Omega^-(2d,q)$;

PROOF: Firstly, consider $(t^v)^\delta$. This gives a matrix of the following form:

$$\begin{pmatrix} 1 & \omega^{-2} & 0 & 0 & 0 & \dots & 0 & * & * \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & * & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix},$$

where the asterisks represent arbitrary elements of GF(q).

Now we conjugate by t^{v^2}, which only affects the top left 4 × 4 block. A simple calculation shows that this gives the following matrix:

$$B(t) = (t^{v^2})^{-1}(t^v)^\delta t^{v^2} = \begin{pmatrix} 1 & \omega^{-2} & 0 & x & 0 & \dots & 0 & * & * \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & x & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & * & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix},$$

where x is an element of GF(q) and the asterisks represent the same arbitrary elements of GF(q) as in the first step.

We now need to work out how to set the (1, 2) entry to 0. By direct calculation, we can see that conjugating the above matrix by δ^{-1} gives:

$$\begin{pmatrix} 1 & 1 & 0 & y & 0 & \dots & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & y & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & \gamma+\gamma^q & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix},$$

where $y \in$ GF(q) and γ is the primitive element of GF(q^2). As the asterisked entries were not changed by conjugating by t^{v^2}, the portion of the matrix outside the 4×4 block will look like t^v, since $t^{v\delta\delta^{-1}} = t^v$.

Pre-multiplying by t^v then gives:

$$\begin{pmatrix} 1 & 0 & 0 & y & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & y & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

We then write y^{-1} as a polynomial in the primitive element ω with coefficients labelled α_i. Then $(t^v B(t))^{\delta^{i-1}}$ gives the matrix with $y\omega^{-i+1}$ in the (4, 1) entry. Hence, post-multiplying $t^v B(t)$ by $\prod_{i=1}^{e}((t^v B(t))^{\delta^{i-1}})^{\alpha_i}$ gives the matrix that we want.

The other three equations can be shown to hold by a similar method. □

2.10 $\Omega(2d+1,q)$ in its natural representation, q odd

This is done in a similar way to the unitary groups in odd dimension. $\Omega(2d+1,q)$ contains $\Omega^+(2d,q)$ as a subgroup. The generators for $\Omega^+(2d,q)$ are found in terms of the $\Omega(2d+1,q)$ generators and the algorithm proceeds by solving as before. Note that as we are only interested in absolutely irreducible groups, we do not consider the odd dimension orthogonal groups in even characteristic. Such groups are reducible and are isomorphic to $\mathrm{Sp}(2d,q)$.

Lemma 2.10.1 *Let* $B(h) = (h^{v^2})^{-1}(h^v)^{\frac{q-1}{2}}h^{v^2}$ *and* $B'(h) = ((hs)^{v^2})^{-1}(h^v)^{\frac{q-1}{2}}(hs)^{v^2}$, *where* v *and* s *are generators for* $\Omega(2d+1,q)$ *as they appear in the paper of Leedham-Green and O'Brien [4] and* h *is some other generator. For* $t \in \Omega(2d+1,q)$, *let* a *(respectively* a'*) be the (1, d) entry of* $B(t)$ *(respectively* $B'(t)$*), let* $n = -\frac{a}{2}$ *and* $m = -\frac{a'}{2}$. *Then for odd characteristic:*

- $t \in \Omega^+(2d,q)$ *is formed by* $(t^v)^n B(t) \in \Omega(2d+1,q)$;
- $r \in \Omega^+(2d,q)$ *is formed by* $(r^v)^n B(r) \in \Omega(2d+1,q)$;
- $t' \in \Omega^+(2d,q)$ *is formed by* $(t^v)^m B'(t) \in \Omega(2d+1,q)$;
- $r' \in \Omega^+(2d,q)$ *is formed by* $(r^v)^m B'(r) \in \Omega(2d+1,q)$;

PROOF: Consider $(t^v)^{-1}t^j t^v$. A simple calculation shows that this gives a matrix with the top 4×4 block looking like this:

$$\begin{pmatrix} 1 & * & 0 & 2j \\ 0 & 1 & 0 & 0 \\ 0 & -2j & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

It is easy to see that setting $j = \frac{q-1}{2}$ will give the matrix

$$\begin{pmatrix} 1 & b & 0 & -1 & 0 & \dots & 0 & 0 & c \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & a & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix},$$

where $a = \frac{q-1}{2}, c = q-1$ are b is an arbitrary element of GF(q).

We now need to work out how to set a, b and c to 0.

Consider how t^v acts on $B(t)$ on the left.

$$t^v = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 & 0 & 2 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

Pre-multiplying by t^v, adds 1 to a, adds 2 to c and adds $1+b+ac$ to b. So if $n = -\frac{a}{2}$, then $(t^v)^n$ sets a and b to zero. Let $x' = (t^v)^n B(t)$. As $\Omega(2d,q) < \Omega(2d+1,q)$, we know that $x \in \Omega(2d,q)$ is also in $\Omega(2d+1,q)$. Then $x^{-1}x'$ is a matrix that has a top 4×4 block of the following form:

$$\begin{pmatrix} 1 & * & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

As transevections do not exist in the orthogonal groups, the asterisk must be 0. Hence, b is zero and $x = x'$ and we are done.

The other three equations can be shown to hold by a similar method. □

2.10.1 $\mathrm{SO}(2d+1,q)$

The method for solving the problem for $\mathrm{SO}(2d+1,q)$ is the same as that for the other two special orthogonal groups. The only exception is that the extra generator needed is:

$$\sigma = \begin{pmatrix} \omega^b & 0 & 0 \\ 0 & \omega^{-b} & 0 \\ 0 & 0 & I_{2d-3} \end{pmatrix},$$

where ω is the primitive element of GF(q) and b is determined by $q-1=2^a b$, where b is odd.

Chapter 3

Non-natural representations in the defining characteristic

3.1 Introduction

We now consider an algorithm to find an arbitrary element of a classical group in an absolutely irreducible non-natural representation of the natural characteristic as a word in the image of the standard generators. In general the classical groups that we will be looking at are perfect, however, we also consider how to deal with the non-perfect special orthogonal groups. A representation of a classical group G is a homomorphism into $\mathrm{GL}(n,q)$. When either n or q (or both) differ from the dimension and base field of G, the representation is said to be non-natural. The classical groups that we will be considering are: $\mathrm{SL}(d,q), \mathrm{Sp}(d,q), \mathrm{SU}(d,q), \Omega^+(d,q), \Omega^-(d,q)$ and $\Omega^\circ(d,q)$. The images of such classical groups will be isomorphic to G, possibly modulo some subgroup of the scalar matrices if the representation ϕ that is being considered is not faithful, as the kernel of ϕ will be a normal subgroup and the classical groups are simple modulo scalars.

Suppose that we are given the image of the standard generators of G in a non-natural representation and call the group generated by this image E. As we are given this image, this determines the representation, which we denote ϕ. E will be a subgroup of $\mathrm{SL}(n,q')$ and in this chapter we shall assume that q and q' are both powers of the same prime.

Let g be an arbitrary element of $\mathrm{SL}(n, q')$ that we wish to either find in terms of the generating set or decide that it is not in the group E.

The algorithms for each classical group are split into the following stages. Assuming that $g \in E$, the algorithm will go through each of the stages:

1. reduce to the case where every pre-image of $g \in G$ has zeroes in all but the first entry on the top row, i.e g is set equal to gh, where we have h as an SLP in the standard generators for E and every pre-image of gh is of the aforementioned form;

2. reduce g further so that every pre-image in G has zeroes in all but the first entry on the first column, whilst preserving step 1;

3. calculate the action of this reduced g on a certain elementary abelian p-group K;

4. apply the natural representation algorithm to this action to complete the process.

We now proceed by describing how this algorithm works for each classical group and how the algorithm will decide if $g \in E$.

3.2 $\mathrm{SL}(d, q)$ in a non-natural representation

In this section, we will describe how this algorithm works for $G = \mathrm{SL}(d, q)$.

3.2.1 Constructing ϕ, the map from the natural to the non-natural representation

Given the standard copy $G = \mathrm{SL}(d, q)$ and the image under ϕ of each generator of G in E, we can construct the image of any element of G under ϕ in the following way:

1. Consider an arbitrary element h of $G = \mathrm{SL}(d, q)$;

2. Use `SLWordInGen`, the natural representation algorithm, to write h as a word w in the standard generators of G;

3. ϕ is defined by mapping each of the standard generators of G to their given images in E;

4. Evaluate w on the image of the standard generators in E.

Although we have this method for computing the image under ϕ for an arbitrary element of G, in certain circumstances we will be computing the image of certain elements by calculating by hand a short word in the standard generating set for G for a particular $g \in G$ and then evaluating these words on the given image of the generating set in E. This will reduce both the time taken to undertake these algorithms and their complexity.

3.2.2 Reducing the pre-image of g by one dimension

Consider the maximal parabolic subgroup H of $\mathrm{SL}(d,q)$ that fixes the space spanned by the first basis element. Then this subgroup consists of matrices of the following shape:

$$\begin{pmatrix} \det^{-1} & 0 & \dots & 0 \\ * & & & \\ \vdots & & \mathrm{GL}(d-1,q) & \\ * & & & \end{pmatrix},$$

where the asterisks represent arbitrary elements of $\mathrm{GF}(q)$ and $\det^{-1}$ represents the element of $\mathrm{GF}(q)$ needed to make the matrix have determinant 1.

We now map H to E by the aforementioned map ϕ. Denote the image of H in E by H^ϕ. Instead of using the method as described in Section 3.2.1 to compute H^ϕ, we will calculate by hand a short word in the standard generating set of G for each element of H and then evaluating these words on the given image of the generating set in E. The generating set for H is: $\{s^{v^{-1}}, s^{-1}v, t^{v^{-1}}, \delta^{v^{-1}}, (t^s)^{-1}, \delta\}$

As the following lemma shows, H^ϕ acts reducibly on the underlying vector space $(\mathbb{F}_{q'})^n$ due to it being a p-local subgroup.

Lemma 3.2.1 *The p-local matrix group H^ϕ acts reducibly on the vector space on which it acts.*

PROOF: Any p-subgroup of $\mathrm{GL}(n, q')$, where q is some power of p, can be made upper uni-triangular and hence, it fixes a proper non-zero subspace. Therefore, any subgroup of $\mathrm{GL}(n, q')$ that normalises the p-group also normalises the subspace. So H^ϕ affords a non-trivial submodule U of $(\mathbb{F}_{q'})^n$. □

Recall that $g \in \mathrm{SL}(n, q')$ is the element that we wish to find in terms of the generators, let U be any proper H-submodule of the given module and let $W = U^g$. The algorithm replicates the procedure for the natural representation by killing the first row of the pre-image of the matrix g in $\mathrm{SL}(d, q)$, assuming that $g \in E$, as follows.

As H^ϕ is maximal in E and $H^\phi \leq N_E(U) < E$, it follows that $H^\phi = N_E(U)$.

Consider the elementary abelian group K, a subgroup of $\mathrm{SL}(d, q)$, consisting of matrices of the following shape:

$$\begin{pmatrix} 1 & * & \dots & * \\ 0 & & & \\ \vdots & & I_{d-1} & \\ 0 & & & \end{pmatrix},$$

where the asterisks represent arbitrary elements of $\mathrm{GF}(q)$.

For $\mathrm{SL}(d, q)$, K is generated by $(d-1)e$ matrices. Each generator is a transvection with a power w^j of the primitive element of $\mathrm{GF}(q)$ on i-th place of the top row. Here, $i \in \{2, \dots, d\}$ and $j \in \{0, \dots, e-1\}$, where e is the degree of the field. The generating set for K is: $\{t^{\delta^{-(j-1)/2}(sv)^{-i}} : 0 \leq i \leq d-2, j \text{ odd and } 1 \leq j \leq e\} \cup \{O^{\delta^{-(j-1)/2}(sv)^{-i}} : 0 \leq i \leq d-2, j \text{ even and } 1 \leq j \leq e\}$, where O is the transvection with the ω in the (1, 2) position, whose construction is defined in Section 2.2.

Now consider K^ϕ. We want to find an element x of K^ϕ that maps W back to U.

If such an element exists, we will then have $U^{gx} = U$. Hence $gx \in N_{\mathrm{SL}(n,q')}(U)$ and, if $g \in E$, gx is in H meaning that we have killed the top row of the pre-image of g. We then dualise this process so that if $g \in E$, we have killed the first column of the pre-image of g. There already exists an algorithm to provide this x. Written by Ruth Schwingel, it has been dubbed `Ruth2` and will be explained later.

We now consider under what circumstances such an element of K does not exist. Such an element will not exist if $g \in E$ and the pre-image of g in G has a zero in its (1, 1) entry or it could mean that $g \notin E$. Recall that the generator v represents a cycle of $(1, \ldots, d)$ in S_d. If we are in this situation, we instead apply `Ruth2` to $g^{\phi(v)^i}$, where i runs through $\{1, \ldots, d-1\}$ until an element x of K is found such that $U^{g^{\phi(v)^i}x} = U$. If no x is found then $g \notin E$ and the algorithm returns 'false'.

By dualising this process, the algorithm can kill the first column of the pre-image of g. This is done by replacing the generator $(t^s)^{-1}$ of H by t and replacing all instances of t with $(t^s)^{-1}$ and O with $(O^s)^{-1}$ in the above generating set for K. This gives generating sets for the inverse transpose of both H and K. $(H^{-T})^{\phi}$ now affords a new submodule U and once again `Ruth2` is used to find a element of $y \in (K^{-T})^{\phi}$ such that $U^{gy} = U$. We can be sure at this point in the algorithm that, if such a y does not exist, $g \notin E$. This is because the algorithm has already performed a check to make sure that there is not a zero in the (1, 1) place of the pre-image of g. If no such y can be found, the algorithm at this point decides that $g \notin E$ and returns 'false'.

3.2.3 The action of the p-group on the reduced matrix g

Having killed both the first column and row in the pre-image of g, we then consider how this reduced g acts on the p-group K^{ϕ}. Let g now denote the reduced g. We discover what each row of the pre-image of g is in the natural representation by the following method. Let $\{K_i : 1 \leq i \leq d\}$ be a subset of the generating set for K, where K_i runs through the $d-1$ transvections whose first row contains exactly one 1 outside the (1, 1) slot. We have an algorithm `MatrixPGroupWordInGen` that writes any element of K^{ϕ} in

terms of the generating set $\{K_i^{\phi} : 1 \leq i \leq d\}$. We apply this to $K_i^{\phi g}$. Hence, we can map these elements back to the natural representation and so we get the required entries (up to a scalar multiple). Note that, if $g \in E$, at this stage of the algorithm $g^{\phi^{-1}}$ normalises K. We now give an example of how this process works for the natural representation in order to obtain the second row for the pre-image of g. Consider the following equation:

$$K_1^{(g^{\phi^{-1}})} = \begin{pmatrix} \alpha^{-1} & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & A^{-1} & \\ 0 & & & \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & & I & \\ 0 & 0 & & & \end{pmatrix} \begin{pmatrix} \alpha & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & A & \\ 0 & & & \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \alpha^{-1}g_{2,2} & \dots & \alpha^{-1}g_{2,d-1} \\ 0 & & & \\ \vdots & & I & \\ 0 & & & \end{pmatrix} \in K.$$

Here, $g_{i,j}$ represents the (i,j)-th entry of the pre-image $g^{\phi^{-1}}$ of g, and A represents the untouched $(d-1) \times (d-1)$ portion of $g^{\phi^{-1}}$. As outlined here, this conjugation process performed in K produces a matrix that has its second to d-th entries on its top row equal to a fixed multiple (here α^{-1}) of the second row of the pre-image of g. However, at this stage of the algorithm, we have $\{K_i^{\phi g}\}$ as elements of the group $E < \mathrm{SL}(n, q')$. The algorithm proceeds, therefore, by writing elements of K^{ϕ} as words in its generating set using the algorithm `MatrixPGroupWordInGen` and hence they can be mapped back to the natural representation. This algorithm will be discussed later. Hence, by forming $(K_i^{\phi})^g$, we can use this algorithm to map these elements back to the natural representation to discover what a fixed multiple of each row of the pre-image of g is.

In this way, the algorithm constructs a candidate for the pre-image of g. As discussed, this candidate will be a multiple of the pre-image of g. We proceed by calculating the determinant θ of this pre-image of g and subsequently dividing $g^{\phi^{-1}}$ by $\sqrt[d]{\theta}$. As our candidate for $g^{\phi^{-1}}$ is equal to a fixed multiple of $g^{\phi^{-1}}$, $\sqrt[d]{\theta}$ will have a solution in $\mathrm{GF}(q)$

if $g \in E$. If no solution exists, $g \notin E$ and the algorithm returns 'false'. As the d-th root of θ is multivalued, the algorithm may not choose the correct root. Disregarding this for the moment, the problem has been reduced to the natural dimension and so we use `SLWordInGen` to complete the problem. If the word returned by `SLWordInGen` does not evaluate to g then it may be that the wrong d-th root of θ was chosen earlier in the algorithm. It therefore may be necessary to go through all possible d-th roots of θ in GF(q) (at most d) until the correct multiple for the pre-image of g is found. If this part of the algorithm fails for all roots of θ, $g \notin E$ and hence the algorithm returns 'false'.

3.2.4 Worked Example on the Exterior Square

Before we look at the pseudo-code for this algorithm, we will look at an example of how this algorithm works. Let $G = \mathrm{SL}(4,7)$. G has the following standard generators:

$$\delta' = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad u' = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$t' = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad v' = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \end{pmatrix}.$$

Consider the following sequence Q:

$$\delta = \begin{pmatrix} 0 & 3 & 5 & 5 & 0 & 4 \\ 6 & 0 & 0 & 4 & 0 & 3 \\ 2 & 5 & 1 & 0 & 6 & 0 \\ 3 & 0 & 1 & 3 & 4 & 4 \\ 0 & 2 & 0 & 5 & 4 & 2 \\ 6 & 2 & 2 & 3 & 0 & 3 \end{pmatrix} \quad u = \begin{pmatrix} 5 & 1 & 1 & 2 & 2 & 4 \\ 6 & 5 & 4 & 6 & 4 & 4 \\ 2 & 5 & 2 & 4 & 4 & 2 \\ 2 & 3 & 1 & 4 & 6 & 3 \\ 5 & 2 & 3 & 0 & 2 & 4 \\ 1 & 0 & 0 & 3 & 6 & 5 \end{pmatrix}$$

$$t = \begin{pmatrix} 0 & 6 & 5 & 0 & 2 & 2 \\ 6 & 0 & 2 & 4 & 0 & 1 \\ 0 & 0 & 5 & 4 & 5 & 6 \\ 5 & 5 & 0 & 5 & 2 & 3 \\ 0 & 0 & 5 & 5 & 2 & 4 \\ 6 & 6 & 6 & 1 & 5 & 1 \end{pmatrix} \quad v = \begin{pmatrix} 0 & 5 & 5 & 4 & 5 & 5 \\ 6 & 6 & 5 & 6 & 6 & 2 \\ 5 & 1 & 1 & 4 & 2 & 6 \\ 5 & 1 & 4 & 0 & 5 & 1 \\ 1 & 4 & 1 & 2 & 3 & 5 \\ 4 & 2 & 2 & 6 & 4 & 4 \end{pmatrix}.$$

The matrices Q generate a group E isomorphic to PSL(4, 7). Let $\phi : G \rightarrow E$ be the representation that maps each of the standard generators to its corresponding image in Q. Let g be the arbitrary matrix of E that we wish to find in terms of Q. We have chosen a g that is in E so that all stages of the algorithm will be shown:

$$g = \begin{pmatrix} 0 & 5 & 1 & 4 & 0 & 4 \\ 6 & 4 & 0 & 6 & 0 & 6 \\ 6 & 4 & 5 & 5 & 5 & 5 \\ 1 & 1 & 4 & 3 & 2 & 5 \\ 0 & 2 & 2 & 6 & 4 & 3 \\ 1 & 5 & 6 & 4 & 2 & 6 \end{pmatrix}.$$

We are unable to see the pre-image under ϕ in G of this matrix g in practice but, in order to demonstrate how the algorithm works, as operations are applied to g, we will also show how this effects $g^{\phi^{-1}}$. The pre-image of g in G that we cannot see is:

$$g^{\phi^{-1}} = \pm \begin{pmatrix} 5 & 4 & 6 & 3 \\ 0 & 6 & 3 & 3 \\ 1 & 5 & 3 & 4 \\ 3 & 2 & 3 & 4 \end{pmatrix}$$

The variables Q, g, d and q are then passed to the algorithm `SLAltRepWordInGen`. Within the algorithm, the variables are subsequently passed to the function `KillRow`. This function modifies g such that $g^{\phi^{-1}}$ has its top row of the form $(\alpha\ 0\ 0\ 0)$, where $\alpha \in$ GF(7). Let $H <$ SL(4, 7) be the group consisting of matrices of the following form:

$$H = \begin{pmatrix} \det^{-1} & 0 & \dots & 0 \\ * & & & \\ \vdots & & \mathrm{GL}(3,7) & \\ * & & & \end{pmatrix},$$

where the asterisks represent arbitrary elements of GF(7) and $\det^{-1}$ represents the element of GF(7) necessary to make each matrix have determinant 1. We write down SLPs in δ, t, u, and v whose inverse images generate H and evaluate these SLPs to get a generating set X_H for H^ϕ. Since H^ϕ is a 7-local group, it acts reducibly and so in this case the Meat-Axe gives a non-trivial H^ϕ-invariant subspace U of $\mathbb{F}_7^6$ with basis {(1 0 0 6 6 4), (0 1 0 2 3 4), (0 0 1 1 0 6)}. The algorithm then also forms U^g having basis {(1 0 0 1 0 6), (0 1 0 2 3 4), (0 0 1 4 4 1)}. Let $K < \mathrm{SL}(4,7)$ be generated by the following matrices:

$$K_1 = t' = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad K_2 = t'^{(u'v')^2} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$K_3 = t'^{u'v'} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Then K^ϕ is generated by $\{t, t^{(uv)^2}, t^{uv}\}$ and `Ruth2` is applied to the pairs (K^ϕ, U) and (K^ϕ, U^g). As $g \in E$ and the (1, 1) entry of $g^{\phi^{-1}}$ is non-zero, this produces a $y \in K^\phi$ such that $U^{gy} = U$:

$$y = \begin{pmatrix} 6 & 1 & 5 & 3 & 5 & 2 \\ 2 & 5 & 5 & 4 & 5 & 0 \\ 0 & 6 & 6 & 0 & 5 & 1 \\ 1 & 6 & 0 & 3 & 0 & 3 \\ 4 & 1 & 3 & 1 & 4 & 0 \\ 3 & 4 & 0 & 6 & 0 & 3 \end{pmatrix} \quad y^{\phi^{-1}} = \pm \begin{pmatrix} 1 & 2 & 3 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

As y has been returned from `Ruth2`, we also have it as an SLP in the given generators of K^{ϕ} and hence can map it back to discover its pre-image $y^{\phi^{-1}}$. The algorithm now exits the function `KillRow` with a new variable a set equal to gy. Here is a and its pre-image that we cannot see in G.

$$a = gy = \begin{pmatrix} 5 & 1 & 3 & 0 & 2 & 4 \\ 5 & 2 & 1 & 4 & 1 & 6 \\ 0 & 6 & 4 & 0 & 4 & 5 \\ 6 & 0 & 5 & 6 & 3 & 2 \\ 0 & 4 & 6 & 6 & 1 & 1 \\ 4 & 0 & 2 & 3 & 5 & 3 \end{pmatrix} \quad a^{\phi^{-1}} = (gy)^{\phi^{-1}} = \pm \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 3 & 3 \\ 1 & 0 & 6 & 2 \\ 3 & 1 & 5 & 5 \end{pmatrix}.$$

It can be seen that $a^{\phi^{-1}}$ has its top row reduced the required form. Q, a, d and q are then passed to `KillColumn` to dualise the process and produce an element $y' \in K$ that maps a to ay' whose pre-image has both its first column and first row killed. For ease of notation, now let a be ay':

$$a = \begin{pmatrix} 6 & 0 & 3 & 4 & 5 & 4 \\ 3 & 4 & 5 & 0 & 2 & 2 \\ 0 & 6 & 5 & 1 & 4 & 4 \\ 5 & 1 & 6 & 3 & 0 & 1 \\ 5 & 6 & 6 & 3 & 0 & 1 \\ 3 & 1 & 4 & 1 & 2 & 1 \end{pmatrix} \quad a^{\phi^{-1}} = \pm \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 3 & 3 \\ 0 & 0 & 6 & 2 \\ 0 & 1 & 5 & 5 \end{pmatrix}.$$

We now form $\{(K_i^{\phi})^a\}$ with $i \in \{1,2,3\}$. As $g \in E$, a is also in E and $\{(K_i^{\phi})^a\} \subset K^{\phi}$, for all i. This can be shown by demonstrating what is happening in the natural dimension:

$$K_1^{a^{\phi^{-1}}} = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 3 & 2 & 3 \\ 0 & 5 & 5 & 5 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 3 & 3 \\ 0 & 0 & 6 & 2 \\ 0 & 1 & 5 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 2 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Note that (4 2 2) is $5^{-1}(6\ 3\ 3)$. That is to say that the second row of the pre-image of a is encoded as a fixed multiple in the top row of $K_1^{a^{\phi^{-1}}}$ by this conjugation. Similarly, $K_2^{a^{\phi^{-1}}}$ will encode the third row of $a^{\phi^{-1}}$ and $K_3^{a^{\phi^{-1}}}$ will encode the fourth. As `MatrixPGroupWordInGen` returns SLPs in the generators of K^{ϕ}, $K_i^{a^{\phi^{-1}}}$ can be calculated for each i and hence, a candidate for the pre-image of a can be constructed:

$$a' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 4 & 2 & 2 \\ 0 & 0 & 4 & 6 \\ 0 & 3 & 1 & 1 \end{pmatrix}.$$

Note that, this is a fixed multiple (5^{-1}) of the actual pre-image of a. As $g = ay'^{-1}y^{-1}$, and we have the pre-images of y and y' in G, we can construct a candidate for the pre-image of g. As $a' = 5^{-1}a^{\phi^{-1}}$, this means that $g^{\phi^{-1}} = 5a^{\phi^{-1}}(y'^{-1})^{\phi^{-1}}(y^{-1})^{\phi^{-1}}$. Therefore, just as a' is a fixed multiple of the actual pre-image of a, g' is a fixed multiple of the pre-image of g:

$$g' = ay'^{-1}y^{-1} = \begin{pmatrix} 1 & 5 & 4 & 2 \\ 0 & 4 & 2 & 2 \\ 3 & 1 & 2 & 5 \\ 2 & 6 & 2 & 5 \end{pmatrix},$$

which in actual fact is $3g$. Next, the determinant of g' is calculated and this matrix is subsequently divided by the 4-th root of this determinant so that it has determinant 1.

This gives the following matrix, which we shall also denote g':

$$g' = \begin{pmatrix} 2 & 3 & 1 & 4 \\ 0 & 1 & 4 & 4 \\ 6 & 2 & 4 & 3 \\ 4 & 5 & 4 & 3 \end{pmatrix}.$$

`SLWordInGen` is then applied to g' to obtain an SLP w. The algorithm then evaluates w on Q and checks to see if this is equal to the original g. Since $g' = -g$ and E is isomorphic to PSL$(4, 7)$, w evaluates to g and hence the algorithm terminates.

3.2.5 Pseudo-code

The inputs to this algorithm will be:

1. The image of the standard generators in the non-natural representation;
2. An arbitrary element g of SL(n, q');
3. d the dimension of the natural representation;
4. q the size of the field that the matrices in the natural representation are over.

The outputs from this algorithm will be:

1. Either true or false to say whether $g \in E$ or not.
2. If $g \in E$, a word for g written as an SLP in the input generators;

Algorithms by other authors are in production to construct SLPs for the standard generators in terms of a given generating set. Hence, by combining this code with the algorithm outlined here, we can write g as a word in any generating set. See [4] for more details.

When we test for irreducibility in the subsequent algorithms, we do so using an algorithm known as the Meat-Axe. In the following code, `MeatAxe` is understood to

take as input a generating set for a matrix group $H < \mathrm{GL}(n, F)$ and, if the associated FH-module is reducible, the algorithm will return a basis for a non-trivial submodule $U < F^n$. See [5] and Chapter 7 of [3] for more details. We first define the following minor function:

- The function `Evaluate`(word, Q) takes as input an SLP named word and a set of matrices Q and returns the evaluation of word on the set Q.

In order to shorten the length of the routine `SLAltRepWordInGen` below, the reader should note that, whenever the variable 'flag' is set to false from a function called within the routine, the algorithm terminates and returns 'false'.

Algorithm 25: SLAltRepWordInGen(Q, g, d, q)

/* Q is a set of matrices that generates a group E isomorphic to a central quotient of $G = \mathrm{SL}(d, q)$. Q is the image in E of the standard generating set for $\mathrm{SL}(d, q)$. E has dimension n and is over the field with q' elements, where q and q' are powers of the same prime. g is an element of $\mathrm{SL}(n, q')$. If $g \in E$, return true plus a word for g in the generating set Q written as an SLP, else return false. */

1 **begin**

2 $n :=$ the degree of E;

3 $q' :=$ the size of the field over which the matrices of Q are defined;

4 flag, a, x, vpower := KillRow(Q, g, d, q); /* See Algorithm 26 */

5 flag, a, y := KillColumn(Q, a, d, q); /* See Algorithm 27 */

6 $Y := \{(K_i^{\phi})^a : 1 \leq i \leq d-1\} \subset \mathrm{SL}(n, q')$;

7 $\bar{Z} :=$ {MatrixPGroupWordInGen$(y, K^{\phi}) : y \in Y$};

8 Evaluate each word on the generating set of K to get $Z = \{Z_1, \ldots, Z_{d-1}\}$;

9 flag, $a' :=$ PreImageOfA(Z);

10 $g' := a'(y^{-1}x^{-1})^{\phi^{-1}}$;

11 M, word := SLWordInGen(G, g');

12 **if** $M \neq$ identity matrix **then**

13 flag, word := WrongScalar$(g', a'$, vpower);

14 **end**

15 return flag, word;

16 **end**

Algorithm 26: `KillRow`(Q, g, d, q)

```
/* g is an element of SL(n,q'). H < SL(n,q') is parabolic,
   K < SL(n,q') is unipotent and both are defined in Section 3.2.2.
   Return an element h of E such that its pre-image in G has
   zeroes in every place of the top row except the (1, 1) entry,
   together with an element k ∈ K such that gk = h. If no such h
   exists, return 'false'.                                        */
```

1 **begin**

2 Create H, the subgroup of G that fixes the first basis element of the natural vector space $V = \mathbb{F}_q{}^d$;

3 $U :=$ `MeatAxe`(H^ϕ);

4 Define $\langle K_i \rangle = K < G$ as above;

5 $x :=$ `Ruth2`(K^ϕ, U);

6 $y :=$ `Ruth2`(K^ϕ, U^g);

7 vpower := 0;

8 **if** $U^{gyx^{-1}} \neq U$ **then**

9 flag, x, y, g, vpower := `GHasPreImageZero`(Q, g);

10 **end**

11 **if** *not* flag **then**

12 return false;

13 **end**

14 return true, gyx^{-1}, yx^{-1}, vpower;

15 **end**

Algorithm 27: KillColumn(Q, h, d, q)

```
/* h is an element of SL(n, q'). Return an element a of E such that
   its pre-image in G has zeroes in every place of the first column
   except the (1, 1) entry, together an element k ∈ K such that
   hk = a. If no such h exists, return 'false'.                    */
```

1 **begin**

2 Form the inverse transpose of the group H that was used in KillRow;

3 $U :=$ MeatAxe(H^{ϕ});

4 Form the inverse transpose of the group K that was used in KillRow;

5 $x :=$ Ruth2(K^{ϕ}, U);

6 $y :=$ Ruth2(K^{ϕ}, U^{h});

7 **if** $U^{gyx^{-1}} \neq U$ **then**

8 return false;

9 **end**

10 return true, hyx^{-1}, yx^{-1};

11 **end**

Algorithm 28: GHasPreImageZero(g)

/* g is an element of SL(n, q') and has pre-image g' in G. g' is known to either have a 0 in its (1, 1) position or $g \notin E$. If $g \notin E$, return false, else return a modified g such that its pre-image has a non-zero entry in its (1, 1) position and two elements that map g to an element whose pre-image has zeroes in every place of the top row except the (1, 1) entry. */

1 **begin**
2 vpower := 0;
3 $(x, y) = (I_n, I_n)$
4 **while** $U^{gyx^{-1}} \neq U$ **do**
5 **if** vpower = d **then**
6 return false;
7 **end**
8 $g := gv$;
9 vpower := vpower +1;
10 $x :=$ Ruth2(K^ϕ, U);
11 $y :=$ Ruth2(K^ϕ, U^g);
12 **end**
13 return true, x, y, g, vpower;
14 **end**

Algorithm 29: PreImageOfA(Z)

/* Z is a set of matrices belonging to K. They are ordered so that the top row of each matrix corresponds to a row in a candidate for the pre-image of a matrix of G. Return a candidate a'. */

1 **begin**

2 d is the dimension of the matrices in Z;

3 $a' :=$ ZeroMatrix;

4 $a'_{1,1} := 1$;

5 **for** $i \in \{1, \ldots, d-1\}, j \in \{2, \ldots, d\}$ **do**

6 $a'_{i,j} := (Z_i)_{1,j}$;

7 **end**

8 **if** $\mathrm{Det}(a')^{-1/d} \notin \mathrm{GF}(q)$ **then**

9 return false;

10 **end**

11 $a' := a'\mathrm{Det}(a')^{-1/d}$;

12 return true, a';

13 **end**

Algorithm 30: WrongScalar(g', a', vpower)

/* a' is a matrix in the intersection of the stabilisers in SL(n, q') of the two submodules defined in Algorithms 27 and 28 (both called U). g' is a candidate for the pre-image of g in the standard copy of $G = \mathrm{SL}(d, q)$. G, Q and v are global variables. Q is as in Algorithm 25 and v is the standard generator of Q. Either multiply g' by the correct multiple and return 'true' and an SLP for g in Q or deduce that g is not in the group generated by Q and hence return false. */

1 **begin**
2 det := $|a'|$;
3 **if** Evaluate(word, Q)$(g(v^{-1})^{-\text{vpower}})^{-1}$ isn't a scalar **then**
4 return false;
5 **end**
6 $R := \{\xi \in \mathrm{GF}(q) : \xi^d - \det = 0\}$;
7 **for** $j \in R$ **do**
8 element := $R_j^{-1} g$;
9 M, word := SLWordInGen(G, element);
10 **if** Evaluate(word, Q) $= g(v^{-1})^{-\text{vpower}}$ **then**
11 break for loop;
12 **end**
13 **end**
14 flag := (M = identity matrix);
15 return flag, word $(\bar{v}^{-1})^{\text{vpower}}$;
16 **end**

3.2.6 Complexity

Let d be the dimension of the natural representation G and n the dimension of the non-natural representation E. Suppose further that G is written over the field $\mathrm{GF}(p^e)$, p a prime. The first step with a significant cost is to create the image in E of the generators of the parabolic subgroup H of G. This contributes $\mathrm{O}(n^3)$ to the complexity, as the number of generators of H is absolutely bounded. We then do the same for the p-group K. Since K contains $(d-1)e$ generators, the total cost is $\mathrm{O}(n^3de)$.

`Ruth2` is then applied, in a majority of cases twice, but at most d times at a total cost of $\mathrm{O}(ud^4n^3e^3+ud^2n^4e)$. Here, u is the dimension of U and this will vary depending on the representation. As a Las Vegas algorithm, a factor of d can be removed from this complexity. The transpose of the matrices in K are then found at a cost of $\mathrm{O}(n^3de)$. `Ruth2` is then applied twice more at a cost of $\mathrm{O}(ud^3n^3e^3 + udn^4e)$. `MatrixPGroupWordInGen` is then used $d-1$ times at a total cost of $\mathrm{O}(d^2n^4e + d^3n^3e^2)$.

Hence, the total cost is $\mathrm{O}(ud^4n^3e^3 + ud^2n^4e)$.

3.2.7 Testing

The code has been implemented in MAGMA and has been tested on thousands of examples. The following input groups have been considered:

- exterior powers of the natural representation;
- symmetric powers of the natural representation;
- the natural representation tensored with itself twisted by the Frobenius map;
- irreducible sections of the natural representation tensored with its dual (known as the adjoint representation of $\mathrm{SL}(d,q)$).

When forming these cases, the generators are usually conjugated by an arbitrary element of $\mathrm{GL}(n,q')$ in order to make the input matrices as arbitrary-looking as possible.

In each of these cases, the following elements of $\mathrm{GL}(n, q')$ have been tested for membership:

- arbitrary elements of E;
- arbitrary elements of $\mathrm{GL}(n, q)$ that are not in E;
- elements that are in the intersection of the stabilisers in $\mathrm{GL}(n, q')$ of the two H^ϕ-modules but not in E;
- elements that are in the intersection of both the above stabilisers and in the normaliser in $\mathrm{GL}(n, q')$ of K^ϕ but not in E;
- the image of the standard generators in E.

3.3 $\mathrm{Sp}(d, q)$ in a non-natural representation

Solving the problem for the symplectic group uses a similar method to the $\mathrm{SL}(d, q)$ case. Here we highlight the differences from the $\mathrm{SL}(d, q)$ algorithm.

1. The subgroup H of $\mathrm{Sp}(d, q)$ in its natural representation that stabilises the space spanned by the first basis vector of the underlying vector space has the following shape:

$$\begin{pmatrix} \theta & 0 & & \cdots & 0 \\ * & \theta^{-1} & * & \cdots & * \\ * & 0 & & & \\ \vdots & \vdots & & \mathrm{Sp}(d-2, q) & \\ * & 0 & & & \end{pmatrix},$$

   where the asterisks and θ represent arbitrary elements of $\mathrm{GF}(q)$. The generating set for H is $\{s^v, vu, u^v, t^v, \delta^v, x, \delta, x^v\}$;

2. The p-group K is generated by conjugates of t by δ plus conjugates of x^{δ^i} by certain products of the generators s, u and v. The generating set of K is $\{x^{v^2\delta^j(s^u)^m(v^u)^i} : i \in \{0,\ldots,\frac{d}{2}-2\}, j \in \{0,\ldots,e-1\}, m \in \{0,1\}\} \cup \{t^{\delta^j}, t_\omega^{\delta^j} : j \in \{0,\ldots,e-1\}\}$, where t_ω is the transvection with ω in the (1, 2) slot, 1s down the main diagonal and zeroes everywhere else. K will have the following shape:

$$\begin{pmatrix} 1 & * & & \ldots & * \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & I_{d-2} & \\ 0 & * & & & \end{pmatrix},$$

where the asterisks on the top row represent arbitrary elements of GF(q) and the asterisks in the second column represent the uniquely determined elements of GF(q) that are necessary for the matrix to preserve the required symplectic form.

3. As in the SL(d, q) case, there is the possibility that there does not exist such an $x \in K$ that will give $U^{gx} = U$. This will either be due to the pre-image of g having a zero in its (1, 1) entry or it could mean that $g \notin E$. As the generator v only represents a cycle of $(1 \ldots \frac{d}{2})$ in $S_{\frac{d}{2}}$, which cycles the hyperbolic pairs, it is also necessary to consider how the generator s may act on the group. If we are in this situation, we instead apply Ruth2 to $g\phi(v)^i\phi(s)^j$, where i runs through $\{1,\ldots,\frac{d}{2}\}$ and $j \in \{0,1\}$ until either an element x of K is found such that $U^{g\phi(v)^i\phi(s)^j x} = U$ or the algorithm returns 'false'.

4. Having killed the top row of the pre-image of g, we dualise the process to kill the first column of the pre-image of g as for SL(d, q). In the SL case, we dualise by transposing the generating sets of H and K. For the symplectic group we can not transpose H, as this would mean that the generating matrices would not preserve the form. Instead, only the generator $(x^v)^{\phi^{-1}}$ is transposed by replacing it with $x^{v^2u^sv^{-1}}$. So H now has the following form:

$$\begin{pmatrix} \theta & * & & \dots & * \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & \mathrm{Sp}(d-2,q) & \\ 0 & * & & & \end{pmatrix}.$$

5. Finally, towards the end of the algorithm, it is necessary to check that the candidate g' for the pre-image of the arbitary element g in the natural representation is an element of $\mathrm{Sp}(d,q)$. This is done by dividing g' by a suitable field element so that its determinant is 1 and then dividing again to make sure the equation $g'^T J g' = J$ holds, where J is the matrix representing the symplectic form. If the equation doesn't hold, then $g'^T J g'$ will be a multiple of J and so dividing g' by a square root of this multiple will get the equation to hold. This process needs to be carried out because g' needs to be in the symplectic group before it is used as input to `SpWordInGen`, else that algorithm will fail.

3.4 $\mathrm{SU}(d,q)$ in a non-natural representation, d even

This is very similar to the symplectic and SL cases. Once again, we outline the differences.

1. The subgroup H of $\mathrm{SU}(d,q)$ in its natural representation that stabilises the space spanned by the first basis vector of the underlying vector space has the following shape:

$$\begin{pmatrix} \theta & * & & \dots & * \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & \mathrm{SU}(d-2,q) & \\ 0 & * & & & \end{pmatrix}.$$

where the asterisks and θ represent arbitrary elements of GF(q^2).

2. The p-group K is generated by conjugates of t by δ plus conjugates of x^{δ^i} by certain products of the generators s, u and v. The generating set of K is $\{x^{y^{-j}(v^u)^i}, x^{y^{-(j-2)}us(v^u)^i}, x_\omega^{y^{-j}(v^u)^i}, x_\omega^{y^{-(j-2)}us(v^u)^i}, t^{y^{-j}} : i \in \{0,\ldots,\frac{d}{2}-2\}, j \in \{0,\ldots,\frac{e}{2}-1\}\}$, where x_ω is the same as the generator x except it has ω in the (1, 3) slot and $-\omega$ in the (4, 2) slot. K will have the following shape:

$$\begin{pmatrix} 1 & 0 & & \ldots & 0 \\ * & 1 & * & \ldots & * \\ * & 0 & & & \\ \vdots & \vdots & & I_{d-2} & \\ * & 0 & & & \end{pmatrix},$$

where the asterisks on the first column represent arbitrary elements of GF(q^2) and the asterisks in the second row represent the uniquely determined elements of GF(q^2) that are necessary for the matrix to preserve the required unitary form.

3. As in the Sp(d, q) case, there is the possibility that there does not exist such an $x \in K$ that will give $U^{gx} = U$ due to the pre-image of g having a zero in its (1, 1) entry or if $g \notin \mathrm{Stab}_{SL(n,q')}(U)$. Once again, we apply `Ruth2` to $g\phi(v)^i\phi(s)^j$, where i runs through $\{1,\ldots,\frac{d}{2}\}$ and $j \in \{0,1\}$ until an element x of K is found such that $U^{g\phi(v)^i\phi(s)^j x} = U$.

4. Having killed the top row of the pre-image of g, we dualise the process to kill the first column of the pre-image of g as for Sp(d, q). Just as in the symplectic group case we can not transpose H, as this would mean that the generating matrices would not preserve the form. Instead, only the generator $(x^{v^2})^{\phi^{-1}}$ is transposed and so H now has the following form:

$$\begin{pmatrix} \theta & 0 & & \dots & 0 \\ * & \theta^{-1} & * & \dots & * \\ * & 0 & & & \\ \vdots & \vdots & & \mathrm{SU}(d-2,q) & \\ * & 0 & & & \end{pmatrix}.$$

5. At this point in the algorithm we have a matrix A whose pre-image in the natural representation has the following shape:

$$\begin{pmatrix} \theta & 0 & 0 & \dots & 0 \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & \mathrm{SU}(d-2,q) & \\ 0 & 0 & & & \end{pmatrix}.$$

As before, we conjugate elements of the p-group K by A. However, there does not necessarily exist a transvection of $\mathrm{SU}(d,q)$ of the following form:

$$\begin{pmatrix} 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & I_{d-2} & \\ 0 & 0 & & & \end{pmatrix}.$$

Instead we have in the (1, 2) entry $\alpha = \omega^{\frac{q+1}{2}}$, when q is odd, and $\alpha = \sqrt{\omega^{q+1}}$, when q is even. This means that when we conjugate this transvection by A, we need to multiply the result by α^{-1} to get the correct candidate for the second row of the pre-image of A.

6. For the symplectic case, it was necessary to make sure that the candidate for the pre-image of g was in the symplectic group before applying the natural representation algorithm to it. This also needs to be done in the unitary group case. In

order to make sure that the pre-image g' is in the unitary group we perform the following calculation. Firstly, we divide g' by a suitable field element so that its determinant is 1. We now wish to divide g' again by a field element so that the equation $\bar{g'}^T J g' = J$ holds, where J is the matrix representing the unitary form. As before, if the equation doesn't hold, then $\bar{g'}^T J g'$ will be a multiple n of J. Then we need to divide g' by a solution in $\mathbb{F}_{q^2}$ to the polynomial $z^{q+1} - n = 0$. The solution set to this equation is $S = \{ \sqrt[q+1]{n}\omega^{\frac{q-1}{q+1}i} : i \in \{0, \ldots, q\}\}$. Hence, we choose an $s \in S$ such that $g's^{-1}$ has determinant 1.

3.5 SU(d, q) in a non-natural representation, d odd

Here we discuss the differences between the code for odd dimension compared to even dimension.

1. H and K are of the same shape as the H and K as described for when d is even. The generating set of K is $\{\bar{x}^{y^{-j}(v^u)^i}, \bar{x}^{y^{-(j-2)}us(v^u)^i}, \bar{x}_\omega^{y^{-j}(v^u)^i}, \bar{x}_\omega^{y^{-(j-2)}us(v^u)^i}, t^{y^{-j}}, x^{y^{-j}} : i \in \{0, \ldots, \frac{d}{2} - 2\}, j \in \{0, \ldots, \frac{e}{2} - 1\}\}$, where $\bar{x}$ is the generator x as defined for SU$(d-1, q)$ considered as an element of SU(d, q) by embedding it in the top left hand corner of SU(d, q), and $\bar{x}_\omega$ is the same as the element $\bar{x}$ except it has ω in the (1, 3) slot and $-\omega$ in the (4, 2) slot.

2. When we conjugate elements of the p-group by our element of the non-natural representation A, we need to look at how we will calculate the last row of the pre-image of A. To discover this last row, we conjugate the image of the following matrix by A:

$$x^v = \begin{pmatrix} 1 & \alpha & 0 & \dots & 0 & 1 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & & & & \\ \vdots & \vdots & & I_{d-2} & & \\ 0 & 0 & & & & \\ 0 & -1 & & & & \end{pmatrix},$$

where $\alpha = \omega^{\frac{q+1}{2}}$, as defined before with ω being the primitive element of GF(q^2). When we calculate $(x^v)^{A^{\phi^{-1}}}$, we find that this gives us the correct last row as its top row, with the exception of the first two positions. We set these to zero to correct the problem.

Another problem with conjugating the generators of the p-group by A is that the generators do not have a 1 in the appropriate place in the top row of the generator but a power of ω instead. This is due to the fact that we create these elements by conjugating $x^{(y^v)^2}$ by y and us. Hence, by taking the (1, 4) entry of the $x^{(y^v)^2 us}$ and dividing every row where this is applicable by this field element, we will have a true multiple for the pre-image of A.

3. When checking that the candidate for the pre-image of A preserves the form, the matrix J that we use has the following form:

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

3.6 $\Omega^+(d,q)$ in a non-natural representation

This is almost exactly the same as the Symplectic case. Here, we outline the differences between the Symplectic case and this one.

1. The subgroup H of $\Omega^+(d,q)$ in its natural representation that stabilises the space spanned by the first basis vector of the underlying vector space has the following shape:

$$\begin{pmatrix} \theta & * & & \dots & * \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & \Omega^+(d-2,q) & \\ 0 & * & & & \end{pmatrix}.$$

   where the asterisks and θ represent arbitrary elements of $\mathrm{GF}(q)$.

2. As in the $\mathrm{Sp}(d,q)$ case, there is the possibility that there does not exist such an $x \in K$ that will give $U^{gx} = U$ due to either the pre-image of g having a zero in its (1, 1) entry, or it could be because $g \notin E$. Once again, we apply `Ruth2` to $g\phi(v)^i\phi(s)^j$, where i runs through $\{1,\dots,\frac{d}{2}\}$ and $j \in \{0,1\}$ until an element x of K is found such that $U^{g\phi(v)^i\phi(s)^j x} = U$, or the algorithm returns 'false'.

3. Having killed the top row of the pre-image of g, we dualise the process to kill the first column of the pre-image of g as for $\mathrm{Sp}(d,q)$. Just as in the symplectic group case we can not transpose H, as this would mean that the generating matrices would not preserve the form. Instead, only the generator $(x^{v^2})^{\phi^{-1}}$ is transposed and so H now has the following form:

$$\begin{pmatrix} \theta & 0 & & \dots & 0 \\ * & \theta^{-1} & * & \dots & * \\ * & 0 & & & \\ \vdots & \vdots & & \Omega^+(d-2,q) & \\ * & 0 & & & \end{pmatrix}.$$

4. We continue as before so that we get a matrix A whose pre-image in the natural representation has the following shape:

$$\begin{pmatrix} \theta & 0 & 0 & \dots & 0 \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & \Omega^+(d-2,q) & \\ 0 & 0 & & & \end{pmatrix}.$$

5. As before, we conjugate elements of the p-group K by A. However, the orthogonal groups do not contain transvections and hence there does not exist a matrix in $\Omega^+(d,q)$ of the following form:

$$T = \begin{pmatrix} 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & I_{d-2} & \\ 0 & 0 & & & \end{pmatrix}.$$

Instead we conjugate using the image in the non-natural representation of $K_1K_2^{-1}$, where K_1 and K_2 are the following matrices:

$$K_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & 0 & & \\ \vdots & \vdots & \vdots & \vdots & I_{d-4} & \\ 0 & 0 & 0 & 0 & & \end{pmatrix}, K_2 = \begin{pmatrix} 1 & 0 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & 0 & & \\ \vdots & \vdots & \vdots & \vdots & I_{d-4} & \\ 0 & 0 & 0 & 0 & & \end{pmatrix}$$

This gives us a matrix of the following form:

$$K_1 K_2^{-1} = \begin{pmatrix} 1 & 1 & 1 & -1 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & 0 & & \\ \vdots & \vdots & \vdots & \vdots & I_{d-4} & \\ 0 & 0 & 0 & 0 & & \end{pmatrix}.$$

Considering the generators of K as an additive algebra, note that $T = K_1 K_2^{-1} - K_1 + K_2$. Hence, we can discover a multiple of the second row of A at this point by performing `MatrixPGroupWordInGen` on $(K_1 K_2^{-1})^{\phi A}$, $K_1^{\phi A}$ and $K_2^{\phi A}$ and forming the above sum.

6. For the symplectic case, it was necessary to make sure that the candidate for the pre-image of g was in the symplectic group before applying the natural representation algorithm to it. This also needs to be done in for this case. As before, we make sure that the pre-image A' preserves the orthogonal form by dividing g' by a suitable field element so that its determinant is 1, and subsequently dividing A' again by a field element so that the equation $A'^T J A' = J$ holds, where J is the matrix representing the orthogonal form. In addition to this, we also need to make

sure that our candidate for the pre-image of A is in the subgroup $\Omega^+(2d,q)$ and not in the full group $\mathrm{SO}(2d,q)$. We do this by dividing A' by -1, if necessary, to move it to the required coset.

3.7 $\Omega^-(d,q)$ in a non-natural representation

This works in a similar way to the plus type case. Here, we outline the differences between the plus type case and this one.

1. The subgroup H of $\Omega^-(d,q)$ in its natural representation that stabilises the space spanned by the first basis vector of the underlying vector space has the following shape:

$$\begin{pmatrix} \theta & * & & \cdots & * \\ 0 & \theta^{-1} & 0 & \cdots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & \Omega^-(d-2,q) & \\ 0 & * & & & \end{pmatrix}.$$

   where the asterisks and θ represent arbitrary elements of $\mathrm{GF}(q)$.

2. As in the $\Omega^+(d,q)$ case, there is the possibility that there does not exist such an $x \in K$ that will give $U^{gx} = U$, either due to the pre-image of g having a zero in its (1, 1) entry or possibly because $g \notin E$. Once again, we apply `Ruth2` to $g\phi(v)^i\phi(s)^j$, where i runs through $\{1,\ldots,\frac{d}{2}\}$ and $j \in \{0,1\}$ until either an element x of K is found such that $U^{g\phi(v)^i\phi(s)^j x} = U$, or the algorithm returns 'false'.

   Note that we only need to select a column from the first $d-2$ to have a non-zero entry as it is not possible to have a top row of $g^{\phi^{-1}}$ looking like this: $(0,\ldots,0,*,*)$. This is because $\Omega^+(d-2,q)$ sits as a subgroup in this group and the existance of such an element would mean that there exists an element of $\Omega^+(d-2,q)$ with its top row entirely consisting of zeroes.

3. Having killed the top row of the pre-image of g, we dualise the process to kill the first column of the pre-image of g as for $\Omega^+(d,q)$. Just as in the plus type case we can not transpose H, as this would mean that the generating matrices would not preserve the form. Instead, only the generator $(r^v)^{\phi^{-1}}$ is transposed and so H now has the following form:

$$\begin{pmatrix} \theta & 0 & & \dots & 0 \\ * & \theta^{-1} & * & \dots & * \\ * & 0 & & & \\ \vdots & \vdots & & \Omega^-(d-2,q) & \\ * & 0 & & & \end{pmatrix}.$$

4. We continue as before so that we get a matrix A whose pre-image in the natural representation has the following shape:

$$\begin{pmatrix} \theta & 0 & 0 & \dots & 0 \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & \Omega^-(d-2,q) & \\ 0 & 0 & & & \end{pmatrix}.$$

5. As before, we conjugate elements of the p-group K by A. We also have the same problem as in the plus type case that we do not have a transvection with a 1 in the (1, 2) slot. Once again, we can create this element by the same process as the plus case: using the image in the non-natural representation of $K_1K_2^{-1}$, where K_1 and K_2 are as before, and subsequently forming $(K_1K_2^{-1})^{\phi^{-1}(A)} - K_1^{\phi^{-1}(A)} + K_2^{\phi^{-1}(A)}$.

However, the process is not yet finished as for minus type the generators of the p-group are not the same as in plus type. In order to discover the $(d-1)$-th and d-th rows of the pre-image of A, we have used conjugates of t by δ, which has extra entries that will produce unwanted values. For the $(d-1)$-th row, this is easy to overcome: we set the second entry to zero (see Lemma 3.7.1 below). Let

the candidate for the pre-image of A so far be denoted $\hat{A}$. To get the correct d-th row of $\hat{A}$, we first set the second entry of this row to zero as before. We then subtract from the d-th row $t^{v\delta}_{1,d-1}$ times the $(d-1)$-th row and divide the result by $t^{v\delta}_{1,d}$ (see Lemma 3.7.2 below).

Lemma 3.7.1 *Let a be a matrix of shape as shown in point 4 above and let t and v be two of the standard generators for $\Omega^-(d,q)$ as defined in Chapter 2. Then the matrix t^{va} has the top tow (1, θ^{-2}, $\theta^{-1}a_{d-1,3}$, ..., $\theta^{-1}a_{d-1,d}$).*

PROOF: Recall that t^v is of the form:

$$t^v = \begin{pmatrix} 1 & 1 & 0 & \dots & 1 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & & & & \\ \vdots & \vdots & & I_{d-2} & & \\ 0 & 2 & & & & \\ 0 & 0 & & & & \end{pmatrix}.$$

Then a simple calculation shows that $t^v a$ is of the form:

$$t^v a = \begin{pmatrix} \theta & \theta^{-1} & a_{d-1,3} & \dots & a_{d-1,d} \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & A & \\ 0 & * & & & \\ 0 & * & & & \end{pmatrix},$$

where the asterisks are entries in GF(q) and A is the bottom right $d-2 \times d-2$ block of the matrix a. Then another calculation gives rise to the following:

$$t^{va} = \begin{pmatrix} 1 & \theta^{-2} & \theta^{-1}a_{d-1,3} & \dots & \theta^{-1}a_{d-1,d} \\ 0 & * & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & I_{d-2} & \\ 0 & * & & & \\ 0 & * & & & \end{pmatrix},$$

where the asterisks are entries in GF(q). □

Lemma 3.7.2 *Let a be a matrix of shape as shown in point 4 above and let t, δ and v be two of the standard generators for $\Omega^-(d,q)$ as defined in Chapter 2. Let A, B and C be the entries of δas they are defined in Chapter 2. Then the matrix $B^{-1}(t^{v\delta a} - At^{va})$ has the top tow (*, *, $\theta^{-1}a_{d,3}$, ..., $\theta^{-1}a_{d,d}$), where the asterisks represents elements of GF(q).*

PROOF: An easy calculation gives the following:

$$t^{v\delta} = \begin{pmatrix} 1 & 1 & 0 & \dots & A & B \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & & & & \\ \vdots & \vdots & & I_{d-2} & & \\ 0 & 2A & & & & \\ 0 & -2C & & & & \end{pmatrix}.$$

Then post-multiplying by a gives the following:

$$t^{v\delta}a = \begin{pmatrix} \theta & \theta^{-1} & Aa_{3,d-1} + Ba_{3,d} & \dots & Aa_{d,d-1} + Ba_{d,d} \\ 0 & * & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & L & \\ 0 & * & & & \\ 0 & * & & & \end{pmatrix},$$

where the asterisks are entries in GF(q) and L is the bottom right $d-2 \times d-2$ block of the matrix a. Pre-multiplying by a^{-1} then gives:

$$t^{v\delta a} = \begin{pmatrix} 1 & * & \theta^{-1}(Aa_{3,d-1} + Ba_{3,d}) & \dots & \theta^{-1}(Aa_{d,d-1} + Ba_{d,d}) \\ 0 & * & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & I_{d-2} & \\ 0 & * & & & \\ 0 & * & & & \end{pmatrix}.$$

Using the result from the previous lemma, we can see that At^{va} is:

$$t^{va} = \begin{pmatrix} A & A\theta^{-2} & A\theta^{-1}a_{d-1,3} & \dots & A\theta^{-1}a_{d-1,d} \\ 0 & * & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & AI_{d-2} & \\ 0 & * & & & \\ 0 & * & & & \end{pmatrix},$$

and hence $t^{v\delta a} - At^{va}$ is:

$$t^{v\delta a} = \begin{pmatrix} 1-A & * & \theta^{-1}(Ba_{3,d}) & \dots & \theta^{-1}(Ba_{d,d}) \\ 0 & * & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & (1-A)I_{d-2} & \\ 0 & * & & & \\ 0 & * & & & \end{pmatrix}.$$

It can be seen that the result follows by dividing this matrix by B. □

3.8 $\Omega^o(d,q)$ in a non-natural representation

Here we explain the differences between this and the Ω^+ case.

1. The subgroup H of $\Omega^o(d,q)$ in its natural representation that stabilises the space spanned by the first basis vector of the underlying vector space has the following shape:

$$\begin{pmatrix} \theta & * & & \dots & * \\ 0 & \theta^{-1} & 0 & \dots & 0 \\ 0 & * & & & \\ \vdots & \vdots & & \Omega^o(d-2,q) & \\ 0 & * & & & \end{pmatrix}.$$

   where the asterisks and θ represent arbitrary elements of GF(q).

2. Let $\hat{A}$ be the pre-image of A that has been formed from conjugating elements of the p-group by A. In order to obtain a correct multiple of the pre-image, we need to set the $(d, 2)$ entry of $\hat{A}$ to be 0 and divide the entire of the d-th row of $\hat{A}$ by 2.

3. When checking that the candidate for the pre-image of A preserves the form, the matrix J that we use has the following form:

$$J = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & \frac{1}{2} \end{pmatrix}.$$

Chapter 4

Modifying Ruth Schwingel's second algorithm: Ruth2

4.1 Introduction

In 2000, Ruth Schwingel completed her thesis, which contained two algorithms [6]. The second algorithm `Ruth2` takes as input a unipotent matrix group K over a prime field and a subspace U of the natural vector space on which K acts. The algorithm returns the following:

- a canonical element $\bar{U}$ of the orbit of U under K;
- an element $k \in K$ such that $U^k = \bar{U}$;
- generators for the stabiliser of U in K.

In order to use this algorithm to help solve the word problem for classical groups in a non-natural representation, it has to be modified. The following changes were made:

- the input matrix group K can now be over a prime power field;
- the element k would also be returned as an SLP in the generators of K.

The algorithm works by taking each echelonised basis vector of U and using elements of K to kill entries in each vector. The result is another subspace $\bar{U}$, which is minimal with respect to a particular partial ordering, defined later. Informally, this ordering is defined by considering the depth of the basis vectors of U; the depth of a basis vector being the position of its first non-zero element reading from left to right. For the original definition of this partial order, see Section 3.3 of Ruth's thesis [6].

4.2 Extending the algorithm to cope with prime power fields

We now redefine the notion of depth to apply to fields of prime power order. Consider an arbitrary element x of GF(p^e). Then x can be written as a polynomial over ω with coefficients in the field GF(p). GF(p^e) can also be considered as an e-dimensional vector space over GF(p) where the element x can be considered as an e-dimensional vector with entries taken from the coefficients of x when it is considered as a polynomial over ω. For example, in the field GF(5^3), the element $2 + 4\omega + 3\omega^2$ can be considered as the vector (2 4 3).

Definition 4.2.1 *The depth plus of a vector v of length d consists of a pair. The first entry i of the pair is the depth of the vector; that is to say the position of the first non-zero entry of the vector reading from left to right. Suppose that the i-th entry of v is x. If v is over the field GF(p^e), then write x as a vector w of length e. The depth of w is the second entry in the pair. The depth plus of the zero vector is defined to be $(d+1, 1)$.*

For example, say that $v = (0\ 0\ \omega\ 1)$ is over the field GF(5^3). v has a depth plus of (3, 2) as 3 is the depth of v and ω corresponds to the vector (0 1 0), which has depth 2. Using this notion of depth, we can define the partial order for a vector space over a prime power field.

4.2.1 Defining the partial ordering and examples

We repeat the definition of the partial order as it appears in Ruth's thesis. First we note that, given two vectors u and w, we say that $u \circledcirc w$ if the first zero entry in u comes sooner than the first zero entry in w reading from left to right. If the positions of the first non-zero entries in u and w are the same, we move to the next zero entry and so on. If the positions of the non-zero entries of u and w are the same then we say that $u \circledominus w$ with respect to this partial ordering. Furthermore, $u \circledcirc w$ if u contains zero entries and w does not.

Definition 4.2.2 *Given two m-dimensional subspaces U and W of a vector space V with invariant flags $U = U_1 > \ldots > U_m > 0$ and $W = W_1 > \ldots > W_m > 0$ respectively, we say that $U_i \circledcirc W_i$ if one of the following occurs:*

1. *$i = m, U_m = \langle u \rangle, W_m = \langle w \rangle$ and $u \circledcirc w$;*
2. *$i < m$ and $U_{i+1} \circledcirc W_{i+1}$;*
3. *$i < m, U_{i+1} \circledominus W_{i+1}, U_i = \langle U_{i+1}, u \rangle, W_i = \langle W_{i+1}, w \rangle$ and $min_{\circledcirc}\{u + x | x \in U_{i+1}\} \circledcirc min_{\circledcirc}\{w + x | x \in W_{i+1}\}$.*

Definition 4.2.3 *A basis of a subspace is said to be echelonised if:*

1. *The leading non-zero entry of each basis vector is 1.*
2. *No two basis vectors are of the same depth.*
3. *Suppose that we have a basis vector u of greater depth, say d, than another basis vector v. Then v is modified by subtracting a multiple of u from v such that the d-th entry of v becomes zero.*

As an example of how point three above works, if a basis of a subspace U of $\mathbb{F}_q^4$ is given as $\{(1,1,0,0),(0,1,1,1)\}$, then the echelonised basis for U is $\{(1,0,-1,-1),(0,1,1,1)\}$.

Example - making a subspace minimal

Here we show how two subspaces are minimal with respect to this partial ordering. We pay particular attention to prime power fields as there are sufficient examples for prime fields in Ruth's thesis [6]. All examples are over $\mathbb{F}_{7^3}$, with primitive element ω.

1. We first consider when one vector is less than a second. Consider $v = (1, 0, 4, 1 + \omega, 3)$ and $w = (1, 0, 4, 1 + \omega^2, 5)$. Reading from left to right, both vectors are the same until their fourth entry. Considering the fourth entry of each as a vector over the prime field, $v_4 = (1, 1, 0)$ and $w_4 = (1, 0, 1)$. As $w_4 \oslash v_4$, $w \oslash v$.

2. Now consider the following two subspaces of $\mathbb{F}_{7^3}^6$, which have echelonised bases. $U = \langle (1, 4, 0, 0, 1, 0), (0, 0, 1, 0, 2 + \omega, 0), (0, 0, 0, 1, 1, 0) \rangle$ and $W = \langle (1, 3, 0, 0, 6, 0), (0, 0, 1, 0, 3 + 2\omega^2, 0), (0, 0, 0, 1, 1, 0) \rangle$. We define maximal flags by defining $U_1 = U$ and then removing the basis element of minimal depth to create the next subspace down in the flag. Hence, we have $U_2 = \langle (1, 4, 0, 0, 1, 0), (0, 0, 1, 0, 2 + \omega, 0) \rangle, U_3 = \langle (1, 4, 0, 0, 1, 0) \rangle$ and $U_4 = 0$.

   Now, $U_i \ominus W_i$ for $i = 3, 4$, so we look at $i = 2$. Consider the sets $A = \{u + x | x \in U_3\}$ and $B = \{w + x | x \in W_3\}$, where $u = (0, 0, 1, 0, 2 + \omega, 0)$ and $w = (0, 0, 1, 0, 3 + 2\omega^2, 0)$. Then the minimal vectors of A and B are u and w respectively. As $w \oslash u$, $W_2 \oslash U_2$ and hence $W \oslash U$.

4.3 Defining the correct chief series

In order for `Ruth2` to fully canonise the input subspace U, it needs to be able to knock out the maximum possible number of entries in the basis vectors of U. In order to do this, the input unipotent group that is being used to canonise the basis vectors of U must be given a generating set that defines successive terms a specific chief series and this must be the case throughout the algorithm. This chief series must also retain the property that no two generators are of the same *matrix weight*, defined below. An algorithm for calculating this chief series is described in Chapter 5.

Definition 4.3.1 *Let A be an upper unitriangular $d \times d$ matrix. The matrix weight of A is an ordered triple containing positive integers. Index the diagonals above the leading diagonal of A from $I = \{1, \ldots, d-1\}$.*

1. *The first element of the triple is the minimum element $i \in I$ such that i contains a non-zero entry.*
2. *The second entry in the triple is the number of places down diagonal i that the first non-zero entry x lies.*
3. *Considering x as a vector over the prime field $\mathbb{F}_p$, as in the definition of depth plus, the third entry in the triple is the depth of this vector.*

Hence, the matrix weight of:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 2 & \omega & 0 \\ 0 & 1 & 0 & 0 & 3 & 0 & 6 \\ 0 & 0 & 1 & 0 & \omega & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

is (2, 3, 2).

As the algorithm progresses, after each generator $g \in K$ is used to kill an entry in a basis vector, say v of U, it is removed from the generating set. The generators k of K that have the property that the depth plus of $v - vg =$ depth plus of $v - vk$ are then replaced with kg^α, for some integer α so that each $v - vkg^\alpha$ has greater depth plus and hence can be used to kill entries further down in v. Once this process has been done, all entries in the p-group are checked to make sure that no two are of the same matrix weight before the algorithm continues.

4.3.1 Example - canonising a vector over $\mathbb{F}_{7^3}$

In this section, we determine the canonical form of the vector $v_0 = (0, 1, 5+\omega, 4\omega^2)$ over $\mathbb{F}_{7^3}$ under the action of a p-group P generated by the matrices $X = \{g_1, g_2, g_3, g_4, g_5\}$ where

$$g_1 = \begin{pmatrix} 1 & \omega & 2 & 1 \\ 0 & 1 & 5 & 2 \\ 0 & 0 & 1 & \omega^2 \\ 0 & 0 & 0 & 1 \end{pmatrix}, g_2 = \begin{pmatrix} 1 & 0 & 4 & 3 \\ 0 & 1 & 2\omega^2 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, g_3 = \begin{pmatrix} 1 & 0 & 0 & 6\omega \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

$$g_4 = \begin{pmatrix} 1 & 0 & 0 & \omega^2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, g_5 = \begin{pmatrix} 1 & 0 & \omega & 6\omega \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The matrices in X are upper uni-triangular and preserve the required chief series (see Section 5.2 for a definition of this series). We first set $v = v_0$ and determine $(j_0, j_1) = \min\{\texttt{DepthPlus}(vg - v) | g \in X\}$.

$$\begin{aligned} vg_1 = (0, 1, 3+\omega, 5+3\omega^2) &\implies \texttt{DepthPlus}(vg_1 - v) = (3, 1) \\ vg_2 = (0, 1, 5+\omega+2\omega^2, 3+4\omega^2) &\implies \texttt{DepthPlus}(vg_2 - v) = (3, 3) \\ vg_3 = (0, 1, 5+\omega, 1+4\omega^2) &\implies \texttt{DepthPlus}(vg_3 - v) = (4, 1) \\ vg_4 = (0, 1, 5+\omega, 4\omega^2) &\implies \texttt{DepthPlus}(vg_4 - v) = (5, 1) \\ vg_5 = (0, 1, 5+\omega, 4\omega^2) &\implies \texttt{DepthPlus}(vg_5 - v) = (5, 1). \end{aligned}$$

Hence, $(j_0, j_1) = (3, 1)$. Set $x = I_4$. As $j_0 < d - 1 = 5$ and $\texttt{DepthPlus}(vg_1 - v) = (j_0, j_1)$, we set $g = g_1$. We now wish to calculate which power of g will kill the constant term of the third entry of v. This power is 6 as $vg^6 = (0, 1, \omega, 2 + 3\omega^2)$. We now set $v = vg_1^6$ and $x = g_1^6$. There is no further $h \in X$ having weight (3, 1) and so we set $X = \langle g_2, g_3, g_4, g_5 \rangle$ and determine a new (j_0, j_1).

$$vg_2 = (0, 1, \omega + 2\omega^2, 5 + 3\omega^2) \implies \texttt{DepthPlus}(vg_2 - v) = (3, 3)$$
$$vg_3 = (0, 1, \omega, 3 + 3\omega^2) \implies \texttt{DepthPlus}(vg_3 - v) = (4, 1)$$
$$vg_4 = (0, 1, \omega, 2 + 3\omega^2) \implies \texttt{DepthPlus}(vg_4 - v) = (5, 1)$$
$$vg_5 = (0, 1, \omega, 2 + 3\omega^2) \implies \texttt{DepthPlus}(vg_5 - v) = (5, 1).$$

Hence, $(j_0, j_1) = (3, 3)$. As $j_0 < d - 1 = 5$ and $\texttt{DepthPlus}(vg_2 - v) = (j_0, j_1)$, we set $g = g_2$. We now wish to calculate which power of g will kill the constant term of the third entry of v. This power is 0 as $vg = (0, 1, \omega, 2 + 3\omega^2)$. Hence, v and x remain as they are. There is no further $h \in X$ having weight (3, 3) and so we set $X = \langle g_3, g_4, g_5 \rangle$ and see that the new $(j_0, j_1) = (4, 1)$. Set $g = g_3$. We see that the required power of g needed to kill the constant term of the fourth entry of v is 5 since $vg = (0, 1, \omega, 3\omega^2)$. Now, $X = \langle g_4, g_5 \rangle$. However, now $(j_0, j_1) = (5, 1)$ and so the algorithm terminates. Hence, the canonical form of v_0 under the action of the p-group is $(0, 1, \omega, 3\omega^2)$ and $x = g_1^6 g_3^5$ is an element of P that transforms v_0 to its canonical form.

4.3.2 Example - canonising a subspace over $\mathbb{F}_{7^3}$

In this section, we calculate the canonical form of the 2-dimensional subspace U, which has basis $\{(1, \omega, 3 + 3\omega, 5\omega^2), (0, 1, 5 + \omega, 4\omega^2)\}$, under the action of the same p-group as in the previous example. We first determine the canonical form u of the vector $(1, \omega, 3 + 3\omega, 5\omega^2)$ under $P = \langle X \rangle, X = \{g_1, g_2, g_3, g_4, g_5\}$. This was completed in the previous example and we obtained the following:

$$u = (0, 1, \omega, 3\omega^2), \quad x = g_1^6 g_3^5, \quad X = \{g_4, g_5\}.$$

Multiplying each basis element by x we obtain:

$$\{(1,0,3+3\omega,5\omega^2),(0,1,\omega,3\omega^2)\},$$

where the last vector is in canonical form. Then we set up a list *depths* of length dimension of U, containing at its last position the depth of u: *depths* = (,2). The next step is to determine the canonical form under $\langle g_4, g_5\rangle$ of the subspace generated by the next basis vector of U, which is $v = (1,0,3+3\omega,5\omega^2)$, and any other vector that has already been dealt with. Since our subspace has dimension 2, this is the last step in our calculation.

We have $B = \{(1,0,3+3\omega,5\omega^2),(0,1,\omega,3\omega^2)\}$ and we wish to determine the weight of g_4 and g_5 with respect to B.

$$\begin{aligned}\texttt{DepthPlus}(v - vg_4) &= \texttt{DepthsPlus}(0,0,6\omega,\omega) &= (3,2) \notin \mathit{depths}\\ \texttt{DepthPlus}(v - vg_5) &= \texttt{DepthsPlus}(0,0,0,6\omega^2) &= (4,3) \notin \mathit{depths}.\end{aligned}$$

So, we take g_4 as it has minimum weight, and find a suitable power of it to kill the (3, 2) entry of the vector v. We find this power to be 4. Hence we set $v = vg_4^4 = (1,0,1,4+3\omega+3\omega^2)$ and set $x = g_1^6 g_3^5 g_4^4$. We recalculate the weight of g_5 with respect to the modified v.

$$\texttt{DepthPlus}(v - vg_5) = \texttt{DepthsPlus}(0,0,0,6\omega^2) = (4,3) \notin \mathit{depths}.$$

So, we use g_5 to kill the (4, 3) entry of v. The power of g_5 we need is also 4 and so we now have $v = vg_5^4 = (1,0,1,4+3\omega)$ and $x = g_1^6 g_3^5 g_4^4 g_5^4$. At this point, X is empty and so the algorithm terminates.

Hence, the canonical form of $U = \langle(1,\omega,3+3\omega,5\omega^2),(0,1,5+\omega,4\omega^2)\rangle$ under the $P = \langle g_1, g_2, g_3, g_4, g_5\rangle$ is $\langle(0,1,\omega,3\omega^2),(1,0,1,4+3\omega)\rangle$. The stabiliser of this canonical form under P is the trivial group, since X was empty when the algorithm terminated.

The element of P transforming U to its canonical form is $x = g_1^6 g_3^5 g_4^4 g_5^4$.

4.4 Complexity

We now consider the complexity of `Ruth2`. Firstly, we wish to consider its complexity given an arbitrary input. Then we will consider its complexity in the context of its use in this paper, which will be cheaper.

Firstly, a change of basis is applied to the input matrices to make them upper unitriangular using `PInvariantFlag`. This involves multiplying vectors by matrices at a cost of $O(d^2)$ each multiplication. Let X be the generating set for the input group to this part of the algorithm. Let X be of dimension d and let the space that X acts on be of dimension u. At the k-th iteration of the while loop, $u - k$ vectors are multiplied by $|X|$ matrices. This while loop has u iterations. Hence, the number of multiplications is $O(|X|u^2)$ and so the cost of `PInvariantFlag` is $O(|X|u^2d^2)$.

We next use the function `PChiefSeriesGenerators` to give the unipotent group generated by X a generating set that preserves the required chief series (see 5.2 for a definition of this chief series). We first need to consider the function `IncreaseDepthPair`. This runs through each element of X and potentially performs a matrix multiplication and weight calculation for each element. This has a cost of $O(|X|d^3)$. `IncreaseDepthPair` makes repeated calls to the function `FindIncreasePower`, which has negligible complexity. Therefore, the total cost of `IncreaseDepthPair` is $O(|X|d^3)$.

`PChiefSeriesGenerators` calculates the weight of each element of the set X at a cost of $O(d^2 + e)$ each weight calculation. It calls `IncreaseDepthPair` $|X|$ times, hence attributing $O(|X|^2d^3)$ to the total complexity. It both raises each generator of X to a power p and calculates its commutator with every other element of X, at most $|X|$ times. That is a total cost of $O(|X|d^3)$ for the first operation and $O(|X|^2d^3)$ for the second. Hence, the total cost for `PChiefSeriesGenerators` is $O(|X|^2d^3)$.

`Ruth2` next calls `SubspaceCF`. Within `SubspaceCF`, the first function to be called is `VectorCF`. This involves a maximum of d multiplications of a vector by a matrix at a

total cost of $O(d^3)$ and a maximum of $|X|d$ matrix multiplications at a total cost of $O(|X|d^4)$. In order to check that the modified chief series preserves the required form, `PChiefSeriesGenerators` is called again a maximum of $m := \max\{|X|, u\}$ times at a total cost of $O(m|X|^2d^3)$. Hence, the complexity of `VectorCF` is $O(m|X|^2d^3 + |X|d^4)$.

We next look at `NextSubspaceCF`, which is carried out $u-1$ times. One iteration of `NextSubspaceCF` contains a maximum of $|X|d$ matrix multiplications at a total cost of $O(|X|d^4)$ and performs `PChiefSeriesGenerators` a maximum of $m := \max\{|X|, u\}$ times at a total cost of $O(m|X|^2d^3)$. Hence the total cost of `NextSubspaceCF` is $O(m|X|^2d^3 + |X|d^4)$. Hence, the total cost of `SubspaceCF` is $O(m|X|^2d^3u + |X|d^4u)$.

Hence, the total cost of `Ruth2` is $O(m|X|^2d^3u + |X|d^4u)$. At worst, $u = d$ and $|X| = \frac{d(d-1)}{2}e$, meaning that $m = |X|$. This would give an overall complexity of $O(d^{10}e^3)$.

For our algorithm, $|X| = de$, where d is the dimension of the natural representation. u will vary, but the worst it could possibly be is $n/2$. Hence, $m = |X| = de$. Hence, the complexity of `Ruth2` for our specific application is $O(d^3e^3n^4 + (de)n^5)$. In practial applications, however, u is generally at worst d and can be as low as 1. Therefore practically the timings will be considerably faster than this complexity suggests.

4.4.1 Timings

The following table shows a list of timings for various input groups. In each case, the input group is a matrix group of dimension n over a field of size p^e containing $(d-1)e$ generators, and a subspace of $\mathbb{F}_{p^e}^n$ of dimension $d-1$. The input group was constructed by taking the elementary abelian group defined in Section 3.2.2, forming its exterior square and conjugating it by a random element of $\mathrm{GL}(n, q)$. The input subspace is the space U afforded by H^ϕ as also defined in Section 3.2.2. The time taken to construct the inputs is not included in the following timings.

d	n	p	e	Ruth2
4	6	7	1	0.016
5	10	7	1	0.016
6	15	7	1	0.016
7	21	7	1	0.016
8	28	7	1	0.016
9	36	7	1	0.031
10	45	7	1	0.031
11	55	7	1	0.063
12	66	7	1	0.094
13	78	7	1	0.141
14	91	7	1	0.203
15	105	7	1	0.313
16	120	7	1	0.453
17	136	7	1	0.547
18	153	7	1	0.75
19	171	7	1	1.031
20	190	7	1	1.422
21	210	7	1	1.891
22	231	7	1	2.516
19	171	11	1	5.141
19	171	31	1	5.406
19	171	67	1	5.469
19	171	97	1	5.359
12	66	7	1	0.094
12	66	7	2	0.422
12	66	7	3	0.797
12	66	7	4	1.438
12	66	7	5	2.313
12	66	7	6	3.438
12	66	7	7	4.75
12	66	7	8	13.484
12	66	7	9	17.672
12	66	7	10	22.469
12	66	7	11	31.531
12	66	7	12	34.125
12	66	7	13	47.641
12	66	7	14	49.859
12	66	7	15	59.922

Table 4.1: Performance of implementation for a sample of groups

Chapter 5

An algorithm to write an element of any unipotent matrix group as a word in its generating set

5.1 Introduction

Let $K < \mathrm{GL}(d,q)$ be a unipotent matrix group and suppose $Y \in \mathrm{GL}(d,q)$. In this section, we describe an algorithm that decides if Y is in K and, if so, writes Y as an SLP in terms of a given generating set of K. The algorithm will work as follows.

1. A change of basis is applied to Y and the generating set of K to make the matrices upper unitriangular. If $Y \notin K$ then this may not be possible and so the algorithm will return 'false' at this stage. This process is exactly the same as in `Ruth2` and appears in Ruth Schwingel's thesis [6].

2. A new generating set for K is then chosen that defines a specific type of chief series. A description of this chief series and the method of constructing the generating set is described in 5.2.

3. The generators of K are used to kill entries in the matrix Y by considering generators of K of least matrix weight, as defined above, to knock out entries in Y

by multiplication. The process starts at the diagonal above the leading diagonal of Y, and proceeds through that diagonal from the top left to the bottom right. The next diagonal up is dealt with and so on until the generating set of K is exhausted. Y will then either be reduced to the identity element, in which case the algorithm returns an SLP for Y in the generators of K, otherwise the algorithm returns 'false'. See Theorem 5.3.4 for a proof of correctness for this algorithm.

5.2 Finding a suitable chief series (Algorithm 32)

As in Ruth2, we must make sure that the generating set of the input nilpotent group is modified so that it exhibits a chief series. We also require that, at any point in the algorithm, no two generators have the same matrix weight. Unlike in Ruth2, however, once we have made sure that no two generators of K have the same matrix weight initially, we do not need to repeat this process at any other point of the algorithm. This is because, unlike in Ruth2, we do not apply any processes to the matrices that alter their matrix weight.

By applying a change of basis, we make K upper unitriangular and we let its generating set be X. Assuming that X is non-empty, initialise Z to consist of a set of pairs: one for each element of X. The first entry of each pair is an element x of X and the second is the matrix weight of x.

Define a sequence B to be empty. We will have a similar set $\tilde{B}$ for the corresponding SLPs. Let $g = (g_1, g_2) \in Z$ be of least weight and add g_1 to B. Note that g may not necessarily be unique, but this does not matter. We wish to do the following:

- use g_1 to modify Z;
- add g_1^p as a pair with its matrix weight to Z;
- add $[g_1, x]$ as a pair with its matrix weight to Z for all $x \in X$ (or $x \in B$, if this set is smaller).

In more detail, we use the function `IncreaseDepthPair` to modify Z so that for any $z \in Z$ with $z_2 = g_2$, we post-multiply z_1 by a suitable power of g_1 to increase its weight. If this modification results in z_1 becoming the identity, z is removed from Z.

Let p be the characteristic of the field over which we are working. If g_1^p is not the identity, we add this as a pair with its matrix weight to Z.

If the commutator $[g_1, x]$ is non-trivial, for any $x \in X$ (or $x \in B$), then this is also added to Z along with its matrix weight.

Then g is removed from Z.

The algorithm continues with the next element of Z of least matrix weight until Z is empty. B is then returned as the generating set for the matrix group that defines the required chief series.

We remind the reader of the square bracket subscript notation used in `Algorithm` 33. For the (i,j)-th element of a matrix A, this will be denoted $A_{i,j,r}$. If the weight of a matrix is (j_0, j_1, j_2) (see Definition 4.2), then $A_{[j_0,j_1,j_2]}$ denotes the coefficient of ω^{j_2-1} in the (j_0, j_0+j_1)-th entry of the matrix A.

5.2.1 Pseudo-code

Algorithm 31: Initial$(g_1, Z, \bar{Z})$

/* Z is a set of pairs. The first entry of each pair is a matrix and the second is the weight of the matrix in the first entry. $\bar{Z}$ is a list of SLPs, each corresponding to a matrix in Z written as a word in a different set of matrices X. The other input is g_1: a matrix of the same size and over the same field as the elements of Z. Return an extended Z, and corresponding $\bar{Z}$, that generates the same group. */

1 **begin**
2 $v := g_1^p$;
3 **if** $v \neq I_d$ **then**
4 $(Z, \bar{Z}) := (Z \cup (v, \texttt{MatrixWeight}(v)), \bar{Z} \cup \bar{g_1}^p)$;
5 **end**
6 **if** $|X| < |B|$ **then**
7 $(T, \bar{T}) := (X, \bar{X})$;
8 **else**
9 $(T, \bar{T}) := (B, \bar{B})$;
10 **end**
11 **end**
12 **for** $x \in T$ **do**
13 $v := (g_1, x)$;
14 **if** $v \neq I_d$ **then**
15 $(Z, \bar{Z}) := (Z \cup (x, \texttt{MatrixWeight}(x), \bar{Z} \cup \bar{x})$;
16 **end**
17 **end**
18 return Z, $\bar{Z}$;
19 **end**

Algorithm 32: PChiefSeriesGenerators($\bar{X}, \bar{X}$: initial)

/* X is a set of generators generating a matrix group of degree d over a finite field of size q written in upper unitriangular form. $\bar{X}$ is the elements of X written trivially as SLPs in X. Return a modified X, generating the same group, which determines a decreasing chief series for X and the corresponding modified SLPs. The default setting for initial is 'true' and is only set to 'false' when it is used in Ruth2 to recalcuate a chief series for X, having already been set to 'true' once at the start of the algorithm. */

1 **begin**

2 $(B, \bar{B}) := (\emptyset, \emptyset)$;

3 $Z := \{(X_i, \texttt{MatrixWeight}(X_i)) : i \in \{1, \ldots, |X|\}\}$;

4 **while** $|Z| \neq 0$ **do**

5 depth $:= \{g_2 : g \in Z\}$;

6 $(j_0, j_1, j_2) = \min(\text{depth})$;

7 Pick $g \in Z$ such that $\texttt{MatrixWeight}(g_1) = (j_0, j_1, j_2)$, with corresponding $\bar{g} \in \bar{X}$;

8 $(B, \bar{B}) = (B \cup g_1, \bar{B} \cup \bar{g})$;

9 $(Z, \bar{Z}) := \texttt{IncreaseDepthPair}(g, Z, \bar{Z}, (j_0, j_1, j_2))$;

10 **if** initial **then**

11 $(Z, \bar{Z}) := \texttt{Initial}(g_1, Z, \bar{Z})$;

12 **end**

13 **end**

14 return B, $\bar{B}$;

15 **end**

Algorithm 33: IncreaseDepthPair$(g, Z, \bar{Z}, (j_0, j_1, j_2))$

```
/* g ∈ Z is of minimum weight (j0, j1, j2). Modify every element of Z
   such that no element has weight equal to (j0, j1, j2).          */
1  begin
2      (X, X̄) := ∅;
3      for z ∈ Z do
4          if z1 ≠ g1 then
5              if z2 = (j0, j1, j2) then
6                  β := −z[j0,j1,j2] / g[j0,j1,j2];
7                  v := z1 g1^β;
8                  (h, h̄1) := ((v, MatrixWeight(v)), z̄1 ḡ1^β);
9                  else
10                     (h, h̄1) := (z, z̄1);
11                 end
12             end
13             if h ≠ 1 then
14                 (X, X̄) := (X ∪ h, X̄ ∪ h̄);
15             end
16         end
17     end
18     return X, X̄;
19 end
```

5.3 Description of the main algorithm

In this section, we shall describe in detail how the algorithm works and provide a proof of correctness for the algorithm. The method of proof will be based around the ideas in Ruth Schwingel's thesis [6], although the ideas have been modified as the algorithm has

been extended to cope with fields of prime power order. In the next section, we shall produce the algorithm in pseudo-code.

Let $K < \mathrm{GL}(d,q)$ be the input unipotent group and suppose that we have $Y \in \mathrm{GL}(d,q)$. The algorithm defines a vector space $V = \mathbb{F}_q^d$ in order to prepare the input for the algorithm `PInvariantFlag`, which will provide a change of basis matrix C to make K upper unitriangular. We now define a key term for the following theorem.

Definition 5.3.1 *A base for a p-group is an ordered generating set that exhibits a chief series.*

We also require the following lemma.

Lemma 5.3.2 *Let $P = \langle X \rangle$ be a finite p-group, and $Q = \langle Y \rangle^P$ be a normal subgroup of P. Let $g \in Y$, and let $R = \langle \{g^p\} \cup (Y \setminus \{g\}) \cup [g, X] \rangle^P$. Then the index of R in Q is at most p.*

PROOF: Note that $Q = \langle Y \cup [Y,X] \cup [Y,X,X] \cup \ldots \rangle$. Clearly R contains $[Y,X]$, and hence $[Y,X,X]$, $[Y,X,X,X]$ and larger commutators. Also R contains $Y \setminus \{g\}$. So Q/R is cyclic of order at most p, generated by Rg. □

Theorem 5.3.3 *The algorithm* `PChiefSeriesGenerators` *having as input a list X of upper unitriangular $d \times d$ matrices over a field F, and a list $\bar{X}$ of corresponding SLPs, returns a base for the p-group $P = \langle X \rangle$ such that no two matrices have the same matrix weight.*

PROOF: The algorithm starts by setting $Z := \{(x, \texttt{MatrixWeight}(x)) : x \in X\}$ and $B := \emptyset$, with corresponding sets $\bar{Z}$ and $\bar{B}$ set appropriately. `MatrixWeight` here is defined as in Definition 4.2. We create a list *depth* := $\{\texttt{MatrixWeight}(x) : x \in X\}$ and from it choose a minimum element (j_0, j_1, j_2) with respect to the usual lexicographical ordering:

$(a_0, a_1, a_2) < (b_0, b_1, b_2)$, if one of the following holds:

1. $a_0 < b_0$;

2. $a_0 = b_0$ and $a_1 < b_1$;

3. $a_0 = b_0, a_1 = b_1$ and $a_2 < b_2$.

We then enter the while-loop. Let $Z_1 = \{a : (a,b) \in Z\})$. We want to prove that this loop will terminate after finitely many iterations and that, at the end of each iteration, the following induction hypothesis is true: Let $Q = \langle Z_1 \rangle^P$ at the beginning of an iteration and $R = \langle Z_1 \rangle^P$ at the end of the same iteration; then, Q consists of all the elements of P of weight at least (j_0, j_1, j_2), where (j_0, j_1, j_2) is the minimal weight of an element of Q, and R contains the elements of P of weight strictly greater than (j_0, j_1, j_2).

Suppose that our induction hypothesis is true at the end on an iteration and we are about to start the next iteration of the while loop. Then, Q is as above and (j_0, j_1, j_2) is the minimal weight of an element of Q. We choose an $h \in Z$ such that $h_2 = (j_0, j_1, j_2)$. Set $\alpha = h_{[j_0,j_1,j_2]}$ (see Section 2.2.1 for a description of this notation), add h_1 to B and remove h from Z. Now we look for all $z \in Z$ with $z_2 = h_2$. Let $\beta = (z_1)_{[j_0,j_1,j_2]}$ and replace all such $z \in Z$ with the same matrix weight as h with the pair $(h_1 z_1^{-\alpha/\beta}$, $\texttt{MatrixWeight}(h_1 z_1^{-\alpha/\beta}))$.

Next, we see if $h_1^p \neq I_d$. If it is not the identity, we add it to Z along with its matrix weight, also noting that $h_{1[j_0,j_1,j_2]}^p = 0$. We then add to Z every non-trivial commutator $[h_1, x]$ such that $x \in X$ (or $x \in B$, if this set is smaller) along with its matrix weight, noting that $[h_1, x]_{[j_0,j_1,j_2]} = 0, \forall x \in X$. Here, we can take the commutators with elements of B rather than X because $[h_1, X] = [h_1, Z_1 \setminus \{h_1\}] \cup [h_1, B]$ and, because $\langle Z_1 \rangle^P$ contains $Z_1 \setminus \{h_1\}$, it contains $[h_1, Z_1 \setminus \{h_1\}]$. Furthermore, we can take the commutators with elements of B rather than X because $[g_1, X] = [g_1, Z_1 \setminus \{h_1\}] \cup [g_1, B]$ and, because $\langle Z_1 \rangle^P$ contains $Z_1 \setminus \{g_1\}$, it contains $[g_1, Z_1 \setminus \{g_1\}]$.

Hence, Q and R are as they appear in the above lemma and so $|Q : R| \leq p$. However, R consists of elements of weight strictly greater than (j_0, j_1, j_2) and hence $|Q : R| = p$.

The list Z never contains the identity matrix, which has matrix weight $(d, 1, 1)$. Hence, after at most $d(d-1)/2$ iterations, Z is empty and so the while loop terminates. □

We now have the matrices that generate K in upper unitriangular form and they preserve the chief series as outlined above. The same change of basis has been applied to Y to make it upper unitriangular also. We now proceed to kill the entries in Y.

1. Set $\bar{y}$ to be the identity SLP.

2. Define a list *weight* containing a set of pairs. The first element of each pair is a generator of K and the second is its matrix weight. Let (j_0, j_1, j_2) be the minimum matrix weight from this list.

3. We now enter a while loop where each entry of Y will be killed by multiplying Y by elements of K in order to increase its matrix weight until it is the identity matrix. Choose $s \in K$ having matrix weight (j_0, j_1, j_2).

4. Find the power of s needed to kill the (j_0, j_1, j_2) entry (in the matrix weight sense) of Y. This is the $(j_2 - 1)$-th power of the primitive element of the $(j_0, j_0 + j_1)$ entry of Y in the usual sense. This power is $\beta = -(Y_{[j1,j0+j1,j2]})(s_{[j1,j0+j1,j2]})^{-1}$ and we now set $Y = Ys^{\beta}$ and $\bar{y} = \bar{y}s^{\beta}$.

5. s is then removed from Y and its respective weight is removed from *weight*. A new s is chosen of minimal matrix weight and this process is repeated until K is empty.

6. If Y is now the identity element, $\bar{y}^{-1}$ is then returned as the word for Y in the input generating set of K. Otherwise, the algorithm returns false.

Theorem 5.3.4 *Let K be a unipotent matrix group of dimension d over a field F. The algorithm* `MatrixPGroupWordInGen` *having as inputs a generating set for K and an element of $Y \in \mathrm{GL}(d, F)$, either decides that $Y \in K$ and returns a word written as an*

SLP in the generators of K that evaluates in the group to Y, or shows that $Y \notin K$ and hence returns 'false'.

PROOF: Firstly, a change of basis is applied to both Y and K so that they are upper unitriangular, if possible. If Y cannot be made upper unitriangular it is not unipotent, hence not in K and the algorithm returns 'false'. The generators of K are subsequently modified so that they preserve the chief series as returned by `PChiefSeriesGenerators`. Call this new sequence of generators S. As we have already proven, S generates K and so Y is still in K after this change has been made. Let the chief series that S defines be $K = S_1 > S_2 > S_3 > \ldots > S_n = \langle 1 \rangle$

As S forms a chief series, Y will fall into three of the following cases, although we are not necessarily able to tell by inspection which one until the algorithm completes:

1. $Y \notin S_1 \setminus S_2$ because $Y \notin S_1$, and hence not in K. In this case, Y will either have matrix weight less than that of any element in S or contain entries that will not be able to be subsequently killed by successive elements of S. In either case, Y cannot be reduced to the identity.

2. $Y \notin S_1 \setminus S_2$ because $Y \in S_2$. In this case, Y is of greater matrix weight than s_1, where $S = \langle s_1, S_2 \rangle$.

3. $Y \in S_1 \setminus S_2$ and so Y has the same matrix weight as s_1.

Consider the third case. Then there exists an integer m_1 such that $s_1^{m_1} \equiv Y \bmod S_2$ and hence $Y s_1^{-m_1} \in S_2$. Which ever case we are in, we remove the element s_1 of least matrix weight from S and iterate. As at each iteration of the while loop a generator from S is removed, the algorithm terminates. At the end of the algorithm, we will have $w = Y s_1^{-m_1} \ldots s_n^{-m_n}$. If Y were originally in the group, then $w \in S_n$ and hence $w = 1$ and the algorithm returns $s_1^{m_1} \ldots s_n^{m_n}$ as the word for Y. If $w \neq 1$, then $Y \notin S$ and so the algorithm returns false. □

5.4 Pseudo-code

Algorithm 34: FindPower$(s, Y, (j_0, j_1, j_2))$

```
1 ) /* Given two matrices s and Y, find the power β of s such that the
     (j0, j1, j2)-th entry, in the matrix weight sense, of Y s^β is 0.
     Return β and Y s^β                                              */
2 begin
3     β := −(Y_[j1,j0+j1,j2])(s_[j1,j0+j1,j2])^{-1};
4     Y := Y s^β;
5     return β, Y;
6 end
```

Algorithm 35: KillEntries$(Y, S, \bar{S})$

```
/* S is a set of upper unitriangular matrices that preserves a
   chief series.  Y ∈ GL(d,q).  If Y ∈ ⟨S⟩, return w, an SLP
   representing the inverse of Y written as a word in the
   generators of S.  If Y ∉ ⟨S⟩, then Y ≠ I_d and this will be passed
   to algorithm 36 for it to return 'false'.                    */
1  begin
2      w = identity SLP;
3      weight := {(S_i, MatrixWeight(S_i))};
4      (j_0, j_1, j_2) = min(weight);
5      while |S| ≠ 0 do
6          Pick s ∈ S such that MatrixWeight(s) = (j_0, j_1, j_2);
7          s̄ = the corresponding element in S̄;
8          (β, Y) := FindPower(s, Y, (j_0, j_1, j_2));
9          w := w s̄^β;
10         S := S − {s};
11         S̄ := S̄ − {s̄};
12         weight := weight−{(s, s̄)};
13         (j_0, j_1, j_2) = min(weight);
14     end
15     return w, Y;
16 end
```

Algorithm 36: `MatrixPGroupWordInGen`(Y, K)

```
/* K is a nilpotent matrix group of degree d over a finite field of
   size q.  Y ∈ GL(d,q).  If Y ∈ K, return a word for Y in the
   input generating set for K written as an SLP, else return
   'false'.                                                    */
1  begin
2      if |K| = 1 then
3          return the empty word;
4      end
5      d := the degree of K;
6      q := the size of the field over which these matrices are defined;
7      V := (F_q)^d;
8      Find a change of basis matrix C that makes Y and K upper unitriangular or
       return 'false' - see Ruth Schwingel's thesis for more information [6];
9      S := {k^C : k ∈ generators of K};
10     Y := Y^C;
11     S̄ := {trivial SLPs in S corresponding to each element of S};
12     S, S̄ := PChiefSeriesGenerators(S, S̄: initial := true);
13     w, Y := KillEntries(Y, S, S̄);
14     if Y = Id(G) then
15         return true, w^{-1};
16     end
17     return false;
18 end
```

5.4.1 Complexity

First the matrices are made upper unitriangular using `PInvariantFlag` with a cost of $O(|X|d^4)$. Then the group is made into a chief series using `PChiefSeriesGenerators`, which has a cost of $O(|X|^2d^3)$. See Section 4.4 for details.

The while loop in `KillEntries` goes through $|X|$ iterations. In each iteration, it calls `FindPower` once. As `FindPower` only involves one field operation, this adds $|X|$ to the complexity. Hence, the complexity of the algorithm as a whole is $O(|X|d^4+|X|^2d^3)$.

5.4.2 Timings

table shows a list of timings for various input groups. In each case, the input group is a matrix group K of dimension n over a field of size p^e containing $(d-1)e$ generators, together with an element from K. The generating set for K is constructed by the same method as described in 4.4.1 and, as before, the time taken to construct the input is not included in the following timings, which are in seconds.

d	n	p	e	MatrixPGroupWordInGen
10	45	7	1	0.016
10	45	7	2	0.094
10	45	7	3	0.062
10	45	7	4	0.078
10	45	7	5	0.094
10	45	7	6	0.047
10	45	7	7	0.125
10	45	7	8	0.936
10	45	7	9	1.154
10	45	7	10	0.936
10	45	7	11	4.196
10	45	7	12	1.248
10	45	7	13	4.711
10	45	7	14	2.543
10	45	7	15	1.264
10	45	7	16	1.576
10	45	19	1	1.186
10	45	61	1	1.498
10	45	97	1	1.576
4	6	7	5	0
5	10	7	5	0
6	15	7	5	0
7	21	7	5	0.016
8	28	7	5	0.031
9	36	7	5	0.031
10	45	7	5	0.094
11	55	7	5	0.125
12	66	7	5	0.172
13	78	7	5	0.312
14	91	7	5	0.468
15	105	7	5	0.546
16	120	7	5	0.718
17	136	7	5	1.045
18	153	7	5	1.513
19	171	7	5	1.56
20	190	7	5	1.622
21	210	7	5	2.387
22	231	7	5	3.12

Table 5.1: Performance of implementation for a sample of groups

Chapter 6

An implementation

6.1 The Natural Representation

The implementation of these algorithms will be made publicly available in MAGMA.

The computations reported in the following tables were carried out using MAGMA V2.14-2 on a Windows Vista computer with a 2.2GHz AMD Phenom 9500 Quad-Core Processor. We list the CPU time in seconds taken to solve the word problem for each classical group in its natural representation. In each case, the code was run ten times and an average taken of the timings.

d	p	e	SL	Sp	Ω^+	Ω^-
20	7	1	0.016	0	0.016	0.078
20	7	2	0.016	0	0.016	0.047
20	7	3	0.031	0.016	0.016	0.031
20	7	4	0.031	0.016	0.016	0.031
20	7	5	0.047	0.016	0.031	0.031
20	7	6	0.047	0.031	0.047	0.062
20	7	7	0.156	0.125	0.156	0.265
20	7	8	0.078	0.031	0.063	0.25
20	7	9	0.094	0.047	0.078	0.452
20	7	10	0.094	0.031	0.078	0.234
20	7	16	0.172	0.063	0.156	-
20	7	32	0.703	0.172	2.344	-
20	7	64	1.625	0.578	45.594	-
50	11	1	0.094	0.172	0.1	0.234
50	19	1	0.109	0.172	0.1	0.234
50	31	1	0.109	0.172	0.1	0.234
50	41	1	0.109	0.172	0.1	0.234
50	53	1	0.109	0.172	0.1	0.234
50	61	1	0.109	0.172	0.1	0.234
50	97	1	0.109	0.172	0.1	0.234
50	643	1	0.094	0.172	0.1	0.234
50	1063	1	0.094	0.172	0.1	0.234
20	7	10	0.094	0.047	0.078	0.328
50	7	10	0.797	0.359	0.719	1.872
70	7	10	1.8	0.844	1.641	5.85
80	7	10	2.5	1.266	2.328	8.408
90	7	10	3.5	2.016	3.141	9.953
100	7	10	4.5	2.813	4.234	16.037
120	7	10	8	5	7.109	26.099

Table 6.1: Performance of implementation for the natural representation

d	p	e	SU even	d	p	e	SU odd	d	p	e	Ω^0
20	7	2	0.016	21	7	2	0.031	21	7	1	0.016
20	7	4	0.031	21	7	4	0.031	21	7	2	0.031
20	7	6	0.047	21	7	6	0.156	21	7	3	0.031
20	7	8	0.094	21	7	8	0.063	21	7	4	0.031
20	7	10	0.125	21	7	10	0.109	21	7	5	0.047
20	7	12	0.141	21	7	12	0.141	21	7	6	0.063
20	7	14	0.188	21	7	14	0.188	21	7	7	0.156
20	7	16	0.281	21	7	16	0.281	21	7	8	0.172
20	7	18	0.297	21	7	18	0.313	21	7	9	0.234
20	7	20	0.391	21	7	20	0.406	21	7	10	0.219
20	7	32	1.625	21	7	32	2.828	21	7	16	0.422
50	11	2	0.2	51	11	2	0.15	51	11	1	1.6
50	19	2	0.2	51	19	2	0.15	51	19	1	1.8
50	31	2	0.2	51	31	2	0.15	51	31	1	2
50	41	2	0.2	51	41	2	0.15	51	41	1	1.9
50	53	2	0.2	51	53	2	0.15	51	53	1	1.8
50	61	2	0.2	51	61	2	0.15	51	61	1	1.9
50	97	2	0.2	51	97	2	0.15	51	97	1	1.9
20	7	10	0.406	21	7	10	0.109	21	7	10	0.219
50	7	10	0.984	51	7	10	0.875	51	7	10	2.078
70	7	10	2.234	71	7	10	2.047	71	7	10	4.828
80	7	10	3.016	81	7	10	2.922	81	7	10	7.328
90	7	10	4.547	91	7	10	4.031	91	7	10	10.234
100	7	10	5.422	101	7	10	5.531	101	7	10	15.313
120	7	10	9.094	121	7	10	9.609	121	7	10	25.922

Table 6.2: Performance of implementation for the natural representation

6.2 Non-Natural Representations

The computations reported in the following tables were carried out using MAGMA V2.14-2. We list the CPU time in seconds taken to solve the word problem for each classical group in a non-natural representation. For parity across each classical group, we have used a random conjugate in $\mathrm{GL}(d, q)$ of the symmetric square of the natural representation as our input non-natural representation. For some classical groups, such a representation is reducible and hence the algorithm may fail. In these cases, we run the algorithm a number of times until a correct result can be produced and timed.

For all groups here, the times shown are for when membership is being tested on a representation of a group that the algorithm has not seen before. The one exception is for SL, where the column following the SL one shows the timings for when the algorithm has already been performed once on a particular representation E and a second element is now being tested for membership in E. This is because functionality has been added to the SL code so that, once the subgroups H and K of E have been constructed, they do not need to be reconstructed for different elements of the same representation.

d	n	p	e	SL	see above	Sp
4	10	7	1	0.031	0.016	0.031
4	10	7	2	0.047	0.016	0.109
4	10	7	3	0.047	0.047	0.094
4	10	7	4	0.125	0.094	0.109
4	10	7	5	0.187	0.156	0.188
4	10	7	6	0.281	0.265	0.5
4	10	7	7	1.014	0.671	6.594
4	10	7	8	1.685	0.905	1.656
4	10	7	9	1.841	1.451	2.281
4	10	7	10	1.872	1.685	2.141
4	10	7	16	7.176	4.789	6.844
4	10	11	1	0.031	0.031	0.063
4	10	19	1	0.031	0.031	0.047
4	10	31	1	0.031	0.031	0.047
4	10	41	1	0.031	0.031	0.047
4	10	53	1	0.031	0.031	0.063
4	10	61	1	0.031	0.031	0.047
4	10	97	1	0.031	0.031	0.047
5	15	7	1	0.031	0.016	–
6	21	7	1	0.094	0.094	0.078
7	28	7	1	0.094	0.094	–
8	36	7	1	0.218	0.203	0.344
9	45	7	1	0.374	0.328	–
10	55	7	1	0.671	0.577	1.203
11	66	7	1	1.217	1.092	–
12	78	7	1	2.153	1.841	4.297
13	91	7	1	2.699	2.48	–
14	105	7	1	4.727	4.477	12.703
15	120	7	1	8.174	7.847	–
16	136	7	1	10.702	10.156	34.063
17	153	7	1	14.914	14.711	–
18	171	7	1	26.91	22.651	84.516
19	190	7	1	30.795	28.564	–
20	210	7	1	45.24	44.195	191.047
21	231	7	1	80.6	78.765	–
22	253	7	1	81.479	76.83	382.953

Table 6.3: Performance of implementation for the symmetric square

d	n	p	e	SU even	d	n	p	e	SU odd
4	10	7	2	0.063	5	15	7	2	0.203
4	10	7	4	0.125	5	15	7	4	0.546
4	10	7	6	0.609	5	15	7	6	2.137
4	10	7	8	1.625	5	15	7	8	5.195
4	10	7	10	3.781	5	15	7	10	6.989
4	10	7	12	5.563	5	15	7	12	11.357
4	10	7	14	7.313	5	15	7	14	36.301
4	10	7	16	10.234	5	15	7	16	30.857
4	10	11	2	0.078	5	15	11	2	0.234
4	10	19	2	0.063	5	15	19	2	0.25
4	10	31	2	0.094	5	15	31	2	0.25
4	10	41	2	0.078	5	15	41	2	0.281
4	10	53	2	0.094	5	15	53	2	0.281
4	10	61	2	0.078	5	15	61	2	0.343
4	10	97	2	0.094	5	15	97	2	0.343
4	10	7	2	0.063	5	15	7	2	0.234
6	21	7	2	0.594	7	28	7	2	1.451
8	36	7	2	3.906	9	45	7	2	6.006
10	55	7	2	16.734	11	66	7	2	25.272
12	78	7	2	63.969	13	91	7	2	97.033
14	105	7	2	252.141	15	120	7	2	304.249

Table 6.4: Performance of implementation for the symmetric square

d	n	p	e	Ω^+	d	n	p	e	Ω^-	d	n	p	e	Ω^0
6	15	7	1	0.047	6	15	7	1	0.172	5	15	7	1	0.062
6	15	7	2	0.234	6	15	7	2	0.437	5	15	7	2	0.172
6	15	7	3	0.406	6	15	7	3	0.858	5	15	7	3	0.374
6	15	7	4	0.656	6	15	7	4	1.295	5	15	7	4	0.343
6	15	7	5	0.813	6	15	7	5	2.059	5	15	7	5	0.484
6	15	7	6	2.156	6	15	7	6	3.167	5	15	7	6	0.796
6	15	7	7	4.266	6	15	7	7	11.544	5	15	7	7	3.494
6	15	7	8	13.656	6	15	7	8	20.935	5	15	7	8	7.379
6	15	7	9	19.328	6	15	7	9	36.348	5	15	7	9	12.184
6	15	7	10	24.156	6	15	7	10	37.315	5	15	7	10	11.357
6	15	11	1	0.109	6	15	11	1	0.265	5	15	11	1	0.109
6	15	19	1	0.109	6	15	19	1	0.25	5	15	19	1	0.109
6	15	31	1	0.156	6	15	31	1	0.343	5	15	31	1	0.109
6	15	41	1	0.125	6	15	41	1	0.281	5	15	41	1	0.109
6	15	53	1	0.125	6	15	53	1	0.25	5	15	53	1	0.109
6	15	61	1	0.141	6	15	61	1	0.359	5	15	61	1	0.109
6	15	97	1	0.141	6	15	97	1	0.359	5	15	97	1	0.109
6	15	643	1	0.156	6	15	643	1	1.856	5	15	643	1	0.172
6	15	1063	1	0.188	6	15	1063	1	0.515	5	15	1063	1	0.234
8	28	7	1	0.156	8	28	7	1	0.281	7	28	7	1	0.172
10	45	7	1	0.594	10	45	7	1	0.842	9	45	7	1	0.593
12	66	7	1	1.984	12	66	7	1	2.215	11	66	7	1	1.435
14	91	7	1	4.703	14	91	7	1	4.477	13	91	7	1	3.838
16	120	7	1	12.125	16	120	7	1	10.374	15	120	7	1	11.419
18	153	7	1	29.688	18	153	7	1	23.79	17	153	7	1	22.62
20	190	7	1	64.063	20	190	7	1	46.02	19	190	7	1	49.702
22	231	7	1	138.469	22	231	7	1	105.238	21	231	7	1	88.421

Table 6.5: Performance of implementation for the symmetric square

Bibliography

[1] J. J. Cannon, W. Bosma (Eds.) Handbook of Magma Functions, Edition 2.13 (2006), 4350 pages.

[2] Larry C. Grove. Classical Groups and Geometric Algebra.

American Mathematical Society, 2001.

[3] Derek Holt, Bettina Eick and Eamonn A. O'Brien. Handbook of Computational Group Theory.

Chapman and Hall, 2005.

[4] Charles Leedham-Green and Eamonn O'Brien. Constructive Recognition of Classical Groups in Odd Characteristic.

Journal of Algebra v. 322, 833–881, 2009.

[5] Richard Parker, M. Atkinson (ed.). The computer calculation of modular characters (The Meat-Axe).

Computational Group Theory, Academic Press, 267–274, 1984.

[6] Ruth Schwingel. Two Matrix Group Algorithms with Applications to Computing the Automorphism Group of a Finite p-group.

PhD thesis, Queen Mary and Westfield College, University of London, 2000.

[7] Donald E. Taylor. The Geometry of Classical Groups.

Heidermann-Verlag Berlin, 1992.